Operating System/2™ Programming Tools and Information Version 1.2

Dialog Manager Guide and Reference

64F3828

Operating System/2™
Programming Tools
and Information
Version 1.2

Dialog Manager Guide and Reference

First Edition (September 1989)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Note to US Government users - Documentation related to Restricted Rights - Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

About This Book

Who Should Read This Book

This book describes the Dialog Manager (DM) for OS/2®. It is intended to be used by application developers who create dialogs that provide users with an interactive display for entering and processing data.

If you are an experienced application developer, you will find that the Dialog Manager gives you the flexibility and power you need to develop complex, efficient applications. If you are less experienced, you will find that the Dialog Manager provides many built-in functions that allow you to concentrate on specific application requirements, rather than expending effort on lower-level coding concerns.

The Dialog Manager can be used with other OS/2 Programming Tools and Information Version 1.2 applications, such as Presentation Manager™. Application developers who want to use Presentation Manager functions in their applications should consult the *Programming Guide* and the *Presentation Manager Programming Reference* for specific information on that product. For a general overview of adding Presentation Manager functions to DM applications, see Chapter 12, "Additional Features of the Dialog Manager" on page 12-1 in this book.

This book should be used with the *Dialog Tag Language Guide and Reference*, which describes the Dialog Tag Language (DTL) used to define panels and other dialog elements with which DM application users interact.

The windows illustrated in this book are intended to show examples of panels defined using the Dialog Tag Language. They are not intended to be exact representations. The actual window may differ slightly when you display the panels you define using the DTL.

Refer to the *OS/2 Getting Started* book for information about how to name files. Also refer to the "Technical Bulletin" section of the *OS/2 Product Information* book for information regarding an OS/2 Standard Edition Version 1.2 update. This update will contain additional rules for naming files.

How This Book Is Organized

This book is divided into two parts: Dialog Manager Guide and Dialog Manager Reference.

Part 1, "Dialog Manager Guide."
Gives an overview of the Dialog Manager and introduces the various components that make up a DM application. It describes how to use Dialog Manager services to create DM applications and illustrates through examples how to code DM applications to meet your application needs. Read Part 1 to gain an understanding

---

of how Dialog Manager services work together to provide an efficient framework for your applications.

Part 2, "Dialog Manager Reference"
Contains complete reference information for the Dialog Manager services, commands, and system variables. Each service definition contains a syntax diagram, specific descriptions of each parameter used in the service call, a general service description, and examples illustrating how to use the service call in your program code. Once you understand how the features of the Dialog Manager work within your applications, you can use the information in Part 2 to begin coding your DM applications.

Related Publications

The following books contain information that may be useful when you are planning and writing DM applications:

- *Dialog Tag Language Guide and Reference* provides a complete description of the Dialog Tag Language tags and how to prepare Dialog Tag Language source files for use with your DM applications.
- *Dialog Manager and Dialog Tag Language Reference Summary* provides a quick reference for the syntax of Dialog Manager services and Dialog Tag Language tags.
- *Building Programs* provides detailed examples that you may use to model your DM applications and PM applications.
- *Programming Guide* provides a complete description of how to use the Presentation Manager to create PM applications.
- *Presentation Manager Programming Reference* describes the Presentation Manager calls with which your DM application can interact.
- *Programming Overview* defines programming concepts needed to write OS/2 applications.
- *Systems Application Architecture Common User Access Advanced Interface Design Guide*, SC26-4582, describes the advanced interface for software applications so that you can create consistent Operating System/2 (OS/2) applications.
- *Systems Application Architecture Common Programming Interface Presentation Reference*, SC26-4359, describes the interface and protocol on which the OS/2 Presentation Manager is based.
- *Systems Application Architecture Common Programming Interface Dialog Reference*, SC26-4356, defines the interface and protocol on which the OS/2 Dialog Manager is based.

Contents

Part 1. Dialog Manager Guide

Part 1. Dialog Manager Guide

Chapter 1. Introduction to the Dialog Manager

The Dialog Manager (DM) is an OS/2 Presentation Manager (PM) application. It provides a set of dialog services based on underlying PM functions. By calling these services in your application, you can avoid having to learn much of the Presentation Manager interface that provides equivalent function. Figure 1-1 provides a basic overview of how both PM and DM applications interact with the Presentation Manager.

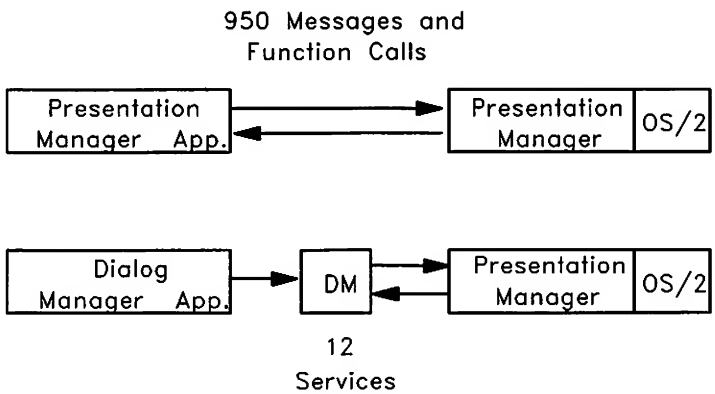


Figure 1-1. How PM and DM Applications Interact with Presentation Manager

The Dialog Manager simplifies and speeds up programming by providing application programming interfaces for panel and message display, variable handling, and controlling application program flow. It uses the Dialog Tag Language (DTL) to build the user interface to your application.

DM eases the conversion from nonprogrammable terminals to programmable workstations by providing you a way to develop graphical user interfaces without learning a new style of programming.

This chapter will help you become more familiar with the capabilities of the Dialog Manager. After reading this chapter, you should be able to begin coding a DM application and coding the related application and help panels. You should also know where to find more information in the related Dialog Manager books.

Systems Application Architecture Compatibility

The Dialog Manager is based on the Systems Application Architecture™ (SAA) Dialog Interface. The Dialog Interface defines elements that are consistent across the SAA environments. SAA applications can be migrated to other application platforms such as the mid-range IBM AS/400™. The OS/2 Dialog Manager contains additional common programming interfaces or *system extensions* that are available only in OS/2. You can design your applications for cross-system use if you avoid using facilities specifically designated as system extensions. Refer to

---

Systems Application Architecture is a trademark of the International Business Machines Corporation.

AS/400 is a trademark of the International Business Machines Corporation.

the *Systems Application Architecture Common Programming Interface Dialog Reference*, SC26-4356, for more information.

The Dialog Manager is a tool used to create applications that conform to the SAA Common User Access (CUA). You do not have to learn all the CUA rules to create applications that conform to CUA. For certain rules that require application involvement, DM provides services that enable the application to enforce those rules.

Language and Application Support

DM-supported languages include IBM C/2™ Version 1.1, IBM COBOL/2™, IBM FORTRAN/2™ Version 1.02, IBM Macro Assembler/2™, IBM Pascal/2™, and Procedures Language 2/Rexx. Hereafter, these languages will be referred to, respectively, as C, COBOL, FORTRAN, MASM, Pascal, and Procedures Language.

DM Application Support

For information about distributing Dialog Manager with your application, see Chapter 11, "Installing Dialog Manager With Your DM Application" on page 11-1.

Designing a Dialog

A dialog is the interaction between a person and a computer. The dialog begins when the application displays a panel and waits for user interaction. The application then may display a series of panels to request more information. The dialog ends when the application ends.

The Dialog Manager displays panels within Presentation Manager windows. Panels are composed of several different *panel elements*, as shown in Figure 1-2.

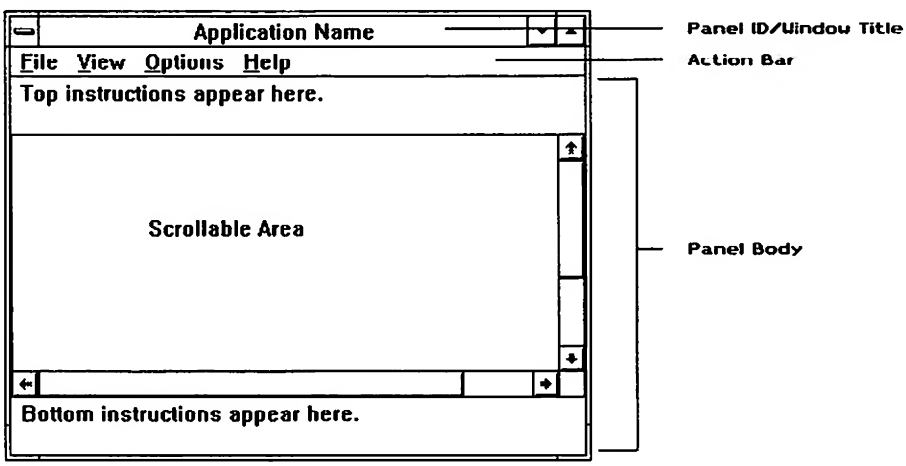


Figure 1-2. Elements of an Application Panel

Many of the panel elements are defined by CUA; others are Dialog Manager extensions to the CUA definition.

C/2, COBOL/2, FORTRAN/2, Macro Assembler/2, and Pascal/2 are trademarks of the International Business Machines Corporation.

The Dialog Manager automatically adds standard window components, including window borders, minimize and maximize icons, system icons, and scroll bars.

The window title of the primary window contains the application name. You can specify optional elements to complete the panel.

The action bar appears directly below the title bar and contains various choices. An action bar pull-down is associated with each choice.

Optional top and bottom instructions guide users on how to proceed with the dialog.

The panel body, which can contain a scrollable area, serves as the main work area and contains various types of fields. These are shown in Figure 1-3.

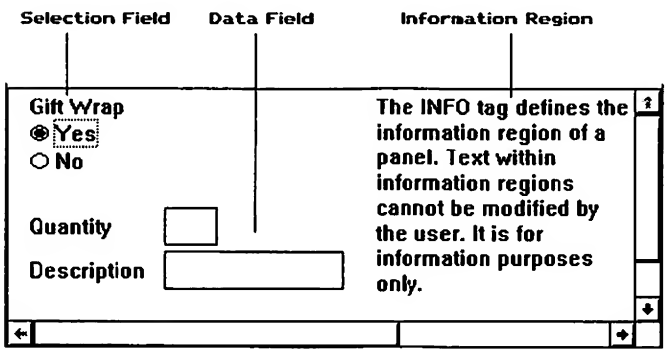


Figure 1-3. Contents of the Scrollable Area

Data fields can be entry fields, where the user types data, or output fields, where a program provides information to the user. The output can be in response to a user request or other action, or a redisplay of previously entered information. Selection fields allow the user to select a choice. You can designate a selection field as a single-choice field (radio buttons), a multiple-choice field (check boxes), or an immediate-action selection field (pushbuttons). Selection lists (list boxes) allow the user to select from a column of choices. List fields are one or more columns of data that can be used to display data and to allow data entry. An information region contains text, such as paragraphs, lists, and notes, that can be used to explain any aspect of the application.

Some panel elements are provided for compatibility with nonprogrammable terminals. You can include a command area as a place the user can enter program-defined and built-in, DM-provided commands that are not available or appropriate in the other parts of the panel. You can also define the function keys used in your application. The function key area (FKA) is located at the bottom of the panel.

You can also verify, range-check, or translate input information entered by the user. DM uses *check lists* to define a list of validity checks to make sure user input is valid. DM uses *translate* and *assignment lists* to translate input into values the program can use.

The application can also display messages to identify input errors or to notify the user of other program events. DM can display a built-in message when input does not pass a validity check.

Users requiring help on how to proceed with the dialog can select help from the action bar pull-down, which lists different kinds of help, or they can select help on a particular field.

A dialog can communicate input information between panels and the application. This is done with variables, which for example, can contain data from an input field and pass it to the application for processing.

Now that you have an idea what the parts of a dialog are, what does the Dialog Manager do to help you design, develop, and run dialogs?

Using the Dialog Manager

The Dialog Manager provides facilities to develop a dialog. You use the Dialog Tag Language (DTL) to design and develop the user interface. You use dialog services and the DM run-time code to develop and run your application.

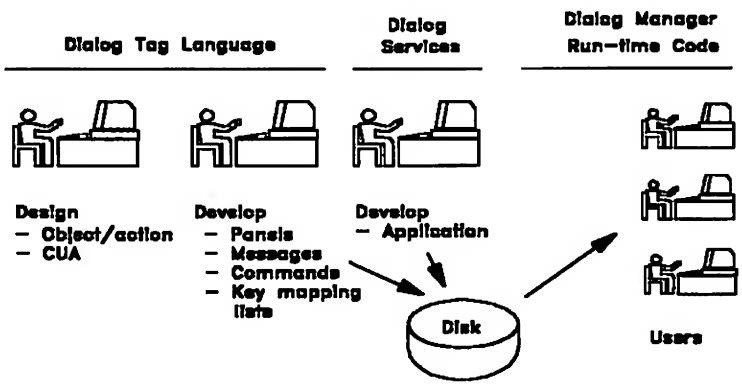


Figure 1-4. Developing DM Applications

The Dialog Manager simplifies program coding by providing the following functions:

- Starts and ends a dialog.
- Displays panels and messages created with the DTL.
- Controls optional features of panel display, such as sounding an alarm.
- Stores and maintains variables that are used by panels and programs.
- Automatically manages the help requests for the application.

Figure 1-4 shows that you begin designing your user interface by using the Dialog Tag Language to create panels, messages, and commands (dialog elements), while keeping in mind the CUA object-action style of user interface design. When you are ready to develop your application, you use dialog services in your program code. Compiled versions of your programs and panels are then stored on disk. When the application runs, the Dialog Manager run-time code presents each panel and handles user interaction.

Dialog Tag Language

The Dialog Tag Language (DTL) is a programming markup language that is based on the Standard Generalized Markup Language (SGML). You use the DTL to define *dialog elements*, which include the following:

Dialog Element	*Function*
Panels	Define both the content and the characteristics of a panel.
Messages	Specify the text of a message.
Command Table	Defines application-supported commands.
Key Mapping Lists	Provide a central place to define keys and their meanings.

Using DTL tags you code, or mark up, dialog elements in source files that are later formatted by the compiler. For example, you use the PANEL tag to begin a panel definition. The PANEL tag looks like this:

```
<panel name=panel1>Application Name
```

Figure 1-5 relates several of the tags to panel elements.

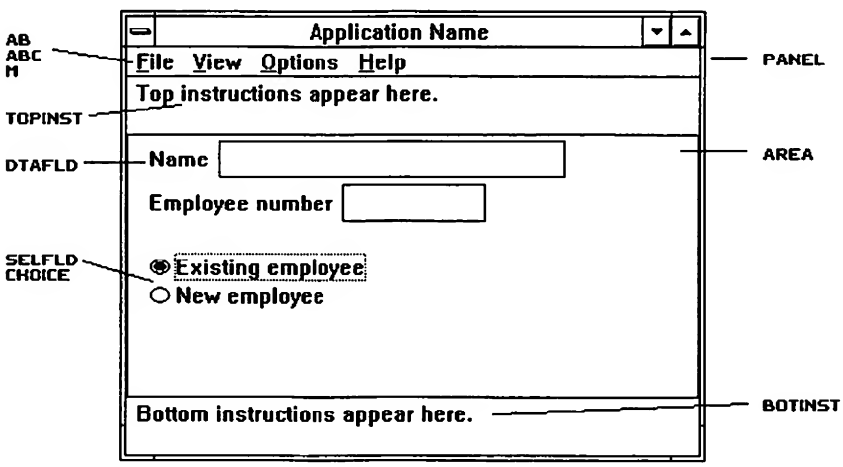


Figure 1-5. Relationship between Tags and Panels

The language concentrates on what a panel element represents and how it relates to other panel elements, rather than solely on its appearance. Dialog elements conform to CUA, both in how they are positioned on a screen and how the user interacts with them.

DTL provides a way to assign variable classes. Variable classes specify what the variable represents, editing operations to be performed, the displayed length, mappings to be applied on input or output, and validation checks.

DTL also provides a utility called DMDISPLY to display panels without running an application. You can display a panel and verify that the panel elements are correct. This allows you to display your user interface and get early feedback from your users.

Dialog Services

To develop your application, you use dialog services in your programming code. Dialog services display panels and messages, manage variables, and control the flow of the dialog. They can be grouped into the following categories:

- Display services
- Variable services
- Control services
- Library service.

Figure 1-6 shows the specific services that belong to each of the categories in the preceding list.

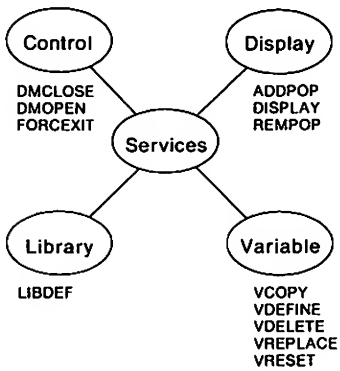


Figure 1-6. Dialog Services

Display services display panels and messages in primary and pop-up windows. Within the context of one display call, many user interactions can occur. The display services process many user interactions, such as moving, sizing, and scrolling, as well as input validation. All of these are provided without any additional program code (other than the DISPLAY call).

Variable services communicate information between the Dialog Manager and the application. Variable services also allow the application to define, copy, and delete variables.

Control services begin and end a DM application and control the flow of a DM application.

The library service allows you to define a set of libraries dynamically.

Relationships between Dialog Elements and Dialog Services

When the application is running, DM uses dialog elements and service calls made by the application to display a panel and then interpret and process user input. For example, when you are designing and coding panels, input and output fields are defined using the DTAFLD tag. The DTAFLD tag contains an attribute called DATAVAR that specifies the variable name associated with the field.

At run time, the application defines this variable to the Dialog Manager using the VDEFINE service. The VDEFINE service allows the application to associate a named variable (specified on the DATAVAR attribute) with a pointer to application storage. The Dialog Manager associates the data variable defined by the application code with the variable defined on the panel and performs any validations or translations that are specified using other DTL tags.

In this example, when control returns to the application, the variable *fname* contains the data entered by the user into the **File name** field.

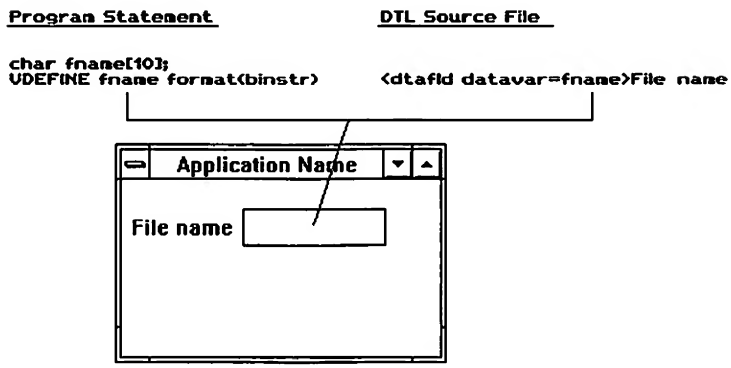


Figure 1-7. Relationship of Variables to Tags

Writing Program Code to Support the Dialog

This section describes how to write an application using the dialog services that the Dialog Manager provides. The order in which these services are discussed is not important, but reflects the order in which you might encounter them in an application.

Two sample DM applications are provided with the Dialog Manager in the OS/2 Programming Tools and Information Version 1.2. One application is written in C language, and the other is written in COBOL. The applications include application panels and help panels defined using the DTL, and the program source code required to run these applications. The OS/2 Programming Tools and Information Version 1.2 will add the Dialog Manager Sample Programs group samples to the Desktop Manager when you choose to install one or both of these sample applications during the OS/2 Programming Tools and Information Version 1.2 installation. To access the installed sample DM applications, double click on one of these programs from the **Group – Dialog Manager Sample Programs** menu.

Developing the Dialog Manager Application

What steps must you take when developing a Dialog Manager application? First, you should review the application requirements and the CUA guidelines to ensure conformance. Remember, to conform to CUA, your application user interface must be designed to support the object-action process sequence. Refer to the *Systems Application Architecture Common User Advanced Interface Design Guide*, SC26-4582, for additional information.

Next, you should identify the application panels and help panels necessary for the application based on what you want the dialog to do. Once these panels are designed, you can begin using a text editor to code the tags necessary to generate the panels. You can also code the tags necessary to generate the other required dialog elements such as key mapping lists, command tables, variable classes, and messages. Once the tags have been coded, they are compiled using the Dialog Tag Language compiler.

You store these compiled files so that they are ready for use by the Dialog Manager. You then code the application to use dialog services.

Controlling the Dialog

Control services establish or end communication with the Dialog Manager run-time code and control the flow of the dialog. Control services include the DMOPEN, DMCLOSE, and FORCEXIT services.

The DMOPEN call establishes communications with the Dialog Manager. It must be the first dialog service in your application. The first DMOPEN service for the application establishes a dialog.

The DMCLOSE service call completes a set of dialog services. The last DMCLOSE call ends the dialog and releases the dialog resources. After the last DMCLOSE, the application cannot issue any service requests to the Dialog Manager unless a new dialog is initiated by DMOPEN.

The FORCEXIT service is used when your application must start a separate thread for a long-running operation. This thread can use the FORCEXIT service call to force the main Dialog Manager thread to return to the application from the current DISPLAY. Refer to the *Programming Overview* for more information on threads.

Displaying Panels

The Dialog Manager provides support for the display of panels. The DISPLAY service call displays a panel and allows user interaction. The user can enter information in entry fields or make selections as defined in the panel definition. The Dialog Manager validates and processes user input according to the panel definition.

Within the context of one DISPLAY service call, the application can use many Dialog Manager services. Many of these services are specified when the panel is generated using the Dialog Tag Language. Others are processed as the user enters data from the workstation. Regardless of where the actual decision is made, much of the processing that occurs when the user interacts with a panel is actually performed by the Dialog Manager without involvement of application code. The Dialog Manager displays panels in Presentation Manager primary and pop-up windows.

The main application window is called the *primary* window and application panels are displayed in this window. Only one application panel at a time can be displayed in the primary window, and each new panel replaces the previous one. The user can move and size primary windows. Every DM application has a primary window.

The application can also display a panel in a *pop-up* window using the ADDPOP and REMPOP services. Use a pop-up window to extend the user's dialog with the primary window, perhaps to ask the user a specific question or to display a message. You can also use another pop-up window on top of the first pop-up window to further extend the dialog. When a pop-up window is on the screen, the user can interact with only the top-level pop-up window. Explicit user action is required to remove the pop-up window. The user can move the pop-up window but cannot size it.

Figure 1-8 on page 1-9 illustrates the use of primary and pop-up windows. In this example, the Dialog Manager DISPLAY, ADDPOP, and REMPOP services are used to display and remove panels in the primary and pop-up windows.

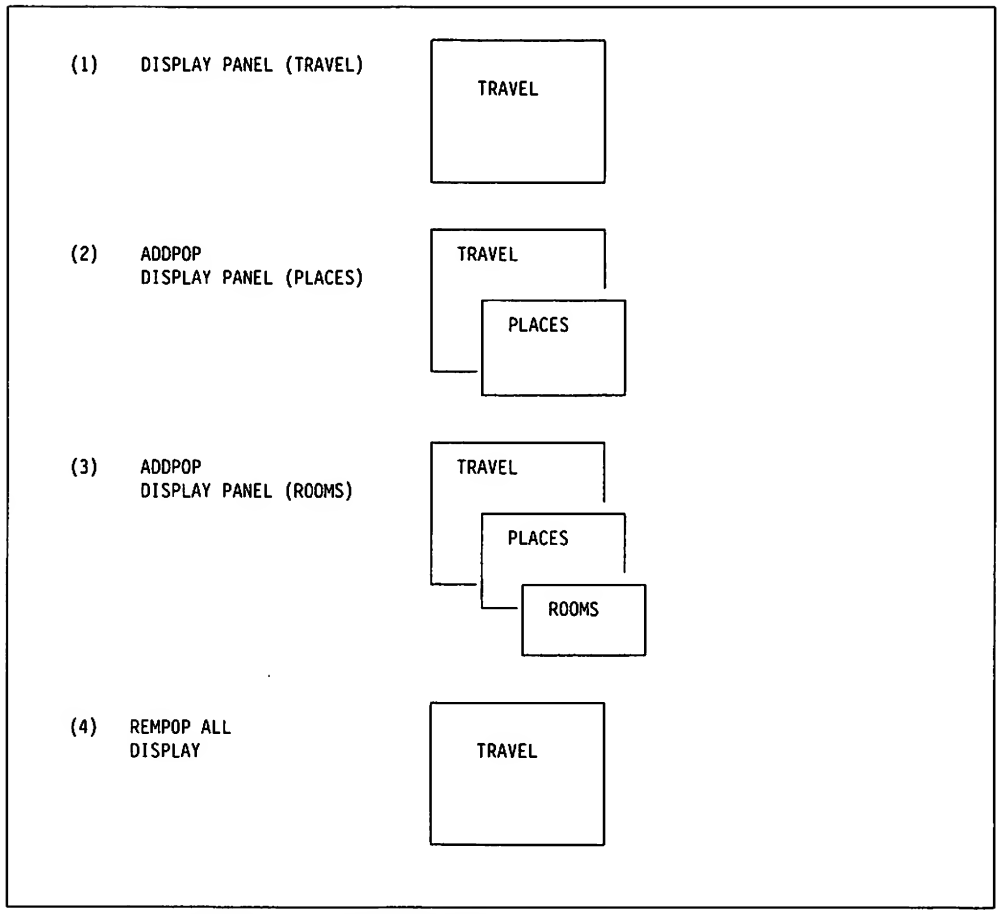


Figure 1-8. ADDPOP and REMPOP Services

Notes:

1. The initial DISPLAY service call specifies that the panel TRAVEL be displayed in the primary window.
2. The ADDPOP and DISPLAY service calls display the PLACES panel in a pop-up window.
3. The next set of ADDPOP and DISPLAY service calls create another pop-up window. This pop-up window displays the ROOMS panel.
4. The REMPOP ALL service call removes the pop-up windows. If you specify the DISPLAY service call without specifying a panel name, the primary window is redisplayed.

Managing Variables

An important concept in the Dialog Manager is that of dialog variables and the dialog variable pool. The dialog variable pool is an area created by Dialog Manager to manage those variables defined and used by an application. The dialog variable pool serves as the data communication vehicle between an application and the Dialog Manager. The variables contained in the dialog variable pool are used by a DM application to get data to and from panels, place data in messages, and dynamically define command actions. For more information, see Chapter 7, "Using Dialog Variables, the Dialog Variable Pool, and Variable Services" on page 7-1.

Variable Services

The Dialog Manager provides a set of services that allow you to define, copy, update, and delete dialog variables. The services are:

- VCOPY (copies a variable)
- VDEFINE (defines a variable)
- VDELETE (deletes the definition of a variable)
- VREPLACE (updates a variable)
- VRESET (resets the dialog variable pool to empty).

System Variables

Dialog Manager provides a set of *system* variables that communicate special information between the DM application and the Dialog Manager. Some of these variables can be modified by the DM application, while others are owned and maintained by the Dialog Manager on behalf of the DM application. All system variable names begin with the letter Z. An example of a system variable is ZAPPLID, which the Dialog Manager sets to the current application ID.

Translating and Validating User Input

DM and DTL provide support for automatically translating and validating users' input. DM uses *translate* and *assignment lists* to translate input and *check lists* to validate input.

Translate lists convert the user's input into values that the program can use. You might use a translate list to minimize keystrokes for the user. For example, you could define a translate list that contains state abbreviations and their full names. You can have users enter just the first character or two of the state's name then translate it into the state's full name.

Translation can occur on input (when the user types a value), and on output (the value stored in the pool is translated before the user sees it).

Assignment lists assign values to one variable based on the content of another variable.

Check lists define a list of validity checks the Dialog Manager applies to values the user enters. You can check to make sure values are valid or fall within a numeric range, or belong to the alphabetic character set. Validation takes place when a user exits a field and before values are passed to the application program.

Translate, assignment, and check lists are defined with DTL tags.

Displaying Messages

The Dialog Manager provides a message facility to communicate information from the DM application to the user in the form of a message.

The Dialog Manager supports three types of messages that the CUA defines:

- Information — indicates to the user that a program is performing normally or has performed normally.
- Warning — is used to notify the user that a potentially undesirable situation could occur.
- Action — is used to notify the user that an exception condition has occurred.

All Dialog Manager messages are displayed in message pop-ups. The user cannot interact with any underlying primary or pop-up windows while the message pop-up is on the screen.

The size of the message pop-up is determined by the Dialog Manager based on the length of the text. The application can specify a position (relative to a field) on the panel in which to display the message pop-up.

Adding Presentation Manager Function to a DM Application

Dialog Manager allows you to provide Presentation Manager functions like graphics and icons in your DM application.

The Dialog Manager provides a way for applications to create their own fields on a Dialog Manager panel. These fields are called *user controls* and they are defined on the panel using the UC tag. You must write code to support a field that is a user control, unlike all other fields on a Dialog Manager panel.

While the Dialog Manager provides a rich set of functions, the application may have requirements that are beyond the scope of Dialog Manager. User exits are points of control that allow the application to gain control during normal processing. The application then has the ability to modify normal Dialog Manager processing or override it.

You must understand Presentation Manager programming to write code to support user controls and user exits. See Chapter 12, "Additional Features of the Dialog Manager" on page 12-1 for more information on user controls and user exits.

Providing Help to Users

The Dialog Manager contains help functions to automatically display help panels and respond to requests for help. This conforms to the Common User Access definition, with some extensions. Help panels are defined with the Dialog Tag Language, and are then compiled.

When you define your application panels, you can provide instructions on completing the panel fields and proceeding in the application by defining these instructions as part of the panel. However, you may want to provide additional instructions that cannot be included easily on the panel or that contain information that not all users will need. You can define help panels that are displayed when the user selects **Help** from the action bar or when the user selects contextual help on a panel field.

For more information on the Dialog Manager help function, see Chapter 9, "Using Dialog Manager Help" on page 9-1.

National Language Support

National language support (NLS) provided by the Dialog Manager and the DTL enables you to translate dialog elements into another national language. The translatable dialog elements include panel, message, command, and key mapping list definitions. You use the DTL to define dialog elements in a source form that is suitable for editing. After translation, users can see and use their national language when interacting with DM applications.

Dialog Manager run-time support includes proper formatting of date and time, decimal and thousands separator characters.

Double-Byte Character Set (DBCS) Support

A single-byte character set is a character set in which one byte represents a character unit and the character set can represent up to 256 different characters. Languages such as Japanese, Chinese, and Korean, however, contain more than 256 different characters. These languages require double-byte character sets (DBCS), in which two adjacent bytes represent a single character unit.

To provide double-byte character set support, the Dialog Manager allows you to specify DBCS and mixed-data formats when defining dialog variables. DBCS specifies that the data is in double-byte format. Mixed-data format specifies that the data includes a mixture of single-byte and double-byte characters.

You can also define DBCS and mixed-data formats for panel fields and panel text. DBCS support also extends to messages. Panel and message text will be properly formatted.

The Dialog Manager ensures proper integrity of both DBCS and mixed-data values. The Dialog Manager performs an even-number-of-bytes check for all usages of DBCS strings. The two-byte code points that represent a double-byte character are never split. If truncation of strings is necessary, DM will ensure the DBCS character code point integrity. Padding for DBCS data is the double-byte blank. Padding for mixed data is always the single-byte blank.

Each DBCS language has a set of characters that cannot be used at the beginning of a line and a set of characters that cannot be used at the end of a line. For example, a right parenthesis, ")," cannot be used at the beginning of a line and a left parenthesis, "(," cannot be used at the end of a line. When formatting text (including multiple-line fields), the Dialog Manager uses these language-dependent character sets and performs the formatting accordingly.

Dialog Manager Code Page Support

The Dialog Manager recognizes and supports different code pages as follows:

- The Dialog Tag Language compiler processes source files encoded using specific code pages. The code page is identified as an option to the compiler. The code page specified for the option is allowed only on the system that supports that option. The code page may be defaulted to a system-provided setting where such a facility is available. (This code page will either identify a single-byte coded character set or a combined single-byte and a double-byte coded character set in a specific encoding scheme.) For more information on code page support, refer to the *OS/2 Programming Reference*.
- When the identified code page allows a mixture of single- and double-byte coded characters, the DTL compiler detects source file transitions from single-byte to double-byte and vice versa.
- Unless specified otherwise, all elements of the Dialog Tag Language syntax are specified using a single-byte coded character set. Specifically, all names of tags and attributes must be represented using single-byte characters. Similarly, all punctuation shown in syntax diagrams must be expressed using single-byte characters.

All tag content can be specified using either single-byte or double-byte characters or a mixture of the two. Tag attribute values may not be encoded using double-byte characters.

- The dialog elements produced by the DTL compiler are internally tagged with the identifier of the code page that was used to prepare the tag source file.
- The Dialog Manager assumes that dialog variables are represented using the scheme identified by the code page of the tag language source. The tags provide a means of specifying that a given dialog variable contains double-byte characters, single-byte characters, or a mixture of the two.

  For correct operation, the code page of the DM application's dialog variables should match the code page specified for the tag source.

Chapter 2. Defining the Flow of a Dialog Manager Application

This chapter describes how to define the flow of a DM application. The topics described are:

- Beginning and ending the DM application
- Creating the primary window
- Creating and displaying pop-up windows.

Beginning and Ending the Dialog Manager Application

Every DM application begins and ends its communication with the Dialog Manager the same way. You initiate communication with the Dialog Manager by calling the DMOPEN service, and you end it by calling the DMCLOSE service. Within these two service calls, you make additional calls to Dialog Manager services to perform the tasks required by your DM application. For example, the DISPLAY, ADDPOP, and REMPOP services allow you to display panels, and create and remove pop-up windows. The variable services (VCOPY, VDEFINE, VDELETE, VREPLACE, and VRESET) allow you to communicate data values to the Dialog Manager.

These services are essential to defining the overall structure and flow of a DM application. Therefore, you should understand how each service works and how they work together before you begin to write a DM application.

DMOPEN Service

The DMOPEN service call initiates communication with the Dialog Manager. The DM application should not try to perform any Dialog Manager service calls unless a successful DMOPEN is completed. One of the parameters that you must specify on the first DMOPEN service call for a DM application is the application identifier, or *application-id*. The application identifier is a 1 – 4 character code that indirectly identifies the dialog elements (panels, messages, command table, key mapping lists) to be used by the DM application. The Dialog Manager reserves all *application-id*s beginning with the characters ISP.

The application ID is used to create the names of the application command table and the application-level library definition file. Using the application ID as a base, the Dialog Manager creates the following names for these files:

*application-id***CMDS.DTL**
: The name of the application command table.

*application-id***LIB.APP**
: The name of the application library definition file.

A DMOPEN service call initializes the DM communication area (*dmcomm*), which establishes a point of control for dialog service calls. The first DMOPEN causes the Dialog Manager to place a unique 8-character identifier in the DM communication area. This identifier is called the *instance-id* because it identifies an instance of a dialog. The *instance-id* contains a combination of the characters A – Z and the numbers 0 – 9. Since all service calls require the *dmcomm* parameter, the Dialog Manager uses the instance ID in the DM communication area to know with which dialog that service call is associated. For more information on defining the DM communication area, see "Service Call Format" on page 4-1.

When your DM application contains only one DMOPEN call, the use of the instance ID is not of much concern. However, because the Dialog Manager does support the concept of multiple DMOPEN calls, an INSTID parameter is provided on the DMOPEN service to associate multiple DMOPEN calls with the same dialog. In order to begin a new dialog you must use the DMOPEN service, without the INSTID parameter, on a separate OS/2 thread. Refer to "Multiple DMOPEN Service Calls" on page 13-1 for more information on multiple DMOPEN calls.

The DMOPEN service also establishes a new dialog variable pool. By issuing multiple DMOPEN calls, programs within the same dialog may have distinct variable pools. To use a single variable pool all programs within the dialog must use the same DM communications area and must not issue multiple DMOPEN calls. If you want to use multiple DMOPEN calls, see "Multiple DMOPEN Service Calls" on page 13-1 for more detailed information.

DMCLOSE Service

A DMCLOSE service call is required for each DMOPEN service call. The DMCLOSE tells the Dialog Manager that the DM application has completed all of its Dialog Manager service calls using that DM communication area. Just as Dialog Manager supports multiple DMOPEN calls, multiple DMCLOSE calls are also supported. The DM communication area is used to link the DMOPEN service call with the corresponding DMCLOSE service call. DMCLOSE will release the dialog variable pool associated with the DM communication area. If it is the last DMCLOSE for the dialog, all resources associated with that dialog are released. After the last DMCLOSE, the application cannot issue dialog service requests without initializing a new dialog using the DMOPEN service call.

For a complete description of multiple DMOPEN and DMCLOSE service calls, see "Multiple DMOPEN Service Calls" on page 13-1.

Creating the Primary Window

A *primary window* is the means by which a DM application conducts a dialog with a user. It is there (or in related pop-up windows) that the Dialog Manager displays panels as requested by the DISPLAY service call. The application's primary window is created automatically on the first DISPLAY service call within a dialog.

The primary window has the following characteristics:

- Only one primary window per dialog is allowed.

  Note: Dialog Manager allows only one dialog per OS/2 thread (and thus per primary window).

- The primary window contains a window title and has a sizeable border.
- The primary window can be moved and sized by the user.

  The window title text used in the primary window is taken from the PANEL tag that is used to define the panel that is displayed in the primary window. This text can be overridden at run time using the ZWINTTL system variable.

  The standard system menu for primary windows is added with the correct options for the panel. Selecting the **Close** choice on a primary window has the same effect as executing the EXIT command. See Chapter 15, "Dialog Manager Commands and Keys" on page 15-1 for a description.

- The initial size of the primary window is determined from the application-defined size of the first panel displayed in the primary window.

The first time (within a dialog) that the DISPLAY service call is issued, the name of a panel must also be specified. This tells the Dialog Manager which panel to display inside the application's primary window. In the process of displaying a panel, the Dialog Manager obtains from the variable pool the current values of dialog variables associated with fields on the panel and places them in the corresponding panel fields. Once the panel has been successfully displayed, the user can interact with the fields on the panel as they were defined using the Dialog Tag Language. When the user has completed interacting with the panel, the DISPLAY service stores the variable values in the pool and returns to the DM application. The same panel can be redisplayed in that window by issuing a DISPLAY service call and not specifying the panel name. In this case, values of variables associated with the panel fields are not obtained from the variable pool. Instead, the values from the previous display are retained. If you want the fields to reflect the current values in the variable pool, issue a DISPLAY service call, specifying the panel name.

You can display another panel in the primary window by issuing another DISPLAY service call, specifying a different panel name. This second DISPLAY call replaces the previous panel in the primary window.

Using Pop-up Windows

Pop-up windows are used to extend the dialog that is taking place in the current window. The following rules apply to pop-up windows:

- A DM application must display a primary window before displaying a pop-up window.
- Users cannot interact with the window from which a displayed pop-up window was called until they complete the interaction with the pop-up window.
- The pop-up window contains a window title bar, which means that the user can move it. The user cannot size the pop-up window.

  The window title text used in the pop-up window is taken from the PANEL tag used to define the panel that is displayed in the pop-up window. If the title is not specified with the PANEL tag, it comes from the primary window. The ZWINTTL system variable cannot be used. The panel definition determines the initial size of the window.

  The system menu for pop-up windows is automatically added. Selecting the **Close** choice on a pop-up window has the same effect as entering the CANCEL command. See Chapter 15, "Dialog Manager Commands and Keys" on page 15-1 for a description.
- The Dialog Manager positions pop-up windows using field-adjacent positioning or offset positioning. See "Window Positioning" on page 2-5 for a description of pop-up window positioning.

The DM application uses the ADDPOP service call to create pop-up windows. After calling the ADDPOP service, the DM application must issue DISPLAY service calls to display panels in that pop-up window. Subsequent DISPLAY calls display the panels in that pop-up window until the DM application issues either a REMPOP service call or another ADDPOP service call.

The REMPOP service call removes the current pop-up window. Following a REMPOP service call, a subsequent DISPLAY service call will display (or redisplay, if no panel name was specified) the panel in the primary window or an underlying pop-up window if ADDPOP service calls were nested. Multiple ADDPOP service calls (together with a DISPLAY service call at each pop-up level) create multiple pop-up windows. The Dialog Manager limits the number of separate pop-up windows that can be created using nested ADDPOP service calls to six.

The optional ALL parameter on the REMPOP service call allows the removal of all pop-up windows created within the current dialog. Without the ALL parameter, the Dialog Manager removes only the top-level pop-up window. If the ALL parameter is not used, the ADDPOP and REMPOP services must be paired.

The following pseudocode example illustrates how to use the DISPLAY, ADDPOP, and REMPOP services to display and remove panels and pop-up windows. See the individual service descriptions for complete syntax and examples.

Example: The following example illustrates how to display a panel in a primary window, display two panels in pop-up windows, and then remove the pop-up windows.

```
DISPLAY MAIN
⋮
ADDPOP
  DISPLAY CLAIM
  ADDPOP
    DISPLAY AUTO
  REMPOP
REMPOP
DISPLAY
```

The first DISPLAY call displays panel MAIN in the primary window. The first pair of ADDPOP and DISPLAY calls display panel CLAIM in a pop-up window. The second ADDPOP and DISPLAY calls display panel AUTO in a pop-up window. The last two REMPOP calls and the last DISPLAY call remove the two pop-up windows and redisplay the panel MAIN. Figure 2-1 shows the results of these DISPLAY and ADDPOP calls.

You can achieve the same results and remove both pop-up windows by specifying REMPOP ALL rather than pairing a REMPOP call with each ADDPOP call. The following example illustrates another way to display and remove the panels:

```
DISPLAY MAIN
⋮
ADDPOP
  DISPLAY CLAIM
  ADDPOP
   DISPLAY AUTO
REMPOP ALL
DISPLAY
```

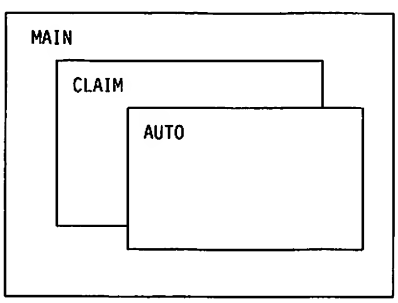


Figure 2-1. Results of DISPLAY and ADDPOP Calls Prior to First REMPOP Call

Window Positioning

This section describes the positioning schemes used by the Dialog Manager to position the primary window and pop-up windows.

The Dialog Manager initially positions the primary window centered on the screen. If the primary window is larger than the screen, it is positioned with the upper-left corner of the window flush with the upper-left corner of the screen. The initial size of the primary window is determined by the size of the first panel displayed in the primary window.

Pop-Up Window Initial Positioning

The Dialog Manager positions pop-up windows using one of the following positioning schemes:

- Field-adjacent positioning
- Offset positioning.

Field-Adjacent Positioning

Using the POPLOC parameter on the ADDPOP service call, pop-up windows can be positioned relative to a particular field on a panel. When the Dialog Manager positions a pop-up window relative to a field, the following algorithm is used:

1. Determine if the pop-up window can be displayed to the right of the field so that no parts of the pop-up window are clipped.
2. If the pop-up window cannot fit to the right of the field, determine if it can be displayed to the left of the field, above the field, or below the field, in that order.
3. If the right or left position is chosen, first attempt to keep the top of the pop-up window even with the top of the field. If that causes the pop-up window to be partially off the screen, then adjust the position vertically to keep the pop-up window completely on the screen.

   If the above or below position is chosen, first attempt to keep the left side of the pop-up window even with the left side of the field. If that causes the pop-up window to be partially off the screen, then adjust the position horizontally to keep the pop-up window completely on the display.
4. If none of the above positions keeps the pop-up window completely visible, and does not overlay the field, the Dialog Manager chooses the position that overlays the least amount of the field. Thus, keeping the pop-up window completely on the screen has higher priority than not overlaying the related field.
5. If a field name that is not defined on the underlying panel is given as the POPLOC parameter, the Dialog Manager returns an error.

Notes:

1. In the field-adjacent positioning algorithm, the output fields include the prompt text and description text (if they exist).
2. The same algorithm defines the field-adjacent positioning scheme used for positioning pop-up windows relative to a field and positioning message pop-up windows.

Offset Positioning

The Dialog Manager offset positioning scheme positions the pop-up window relative to the underlying window. The underlying window could be the DM application primary window or another pop-up window. The offset positioning works the same in both cases. When using offset positioning, the Dialog Manager positions the pop-up window below the underlying window so that the title bar is visible. The pop-up window is positioned to the right of the outside left edge of the underlying window. The fixed distance for horizontal positioning is approximately two character units. The character unit is the width of a zero (0). These fixed distances are used so that when the pop-up windows are offset, they *cascade* evenly, as shown in Figure 2-2 on page 2-7.

Successive pop-up windows are cascaded (down and to the right of the previous window) until the bottom or right edge of the display is reached. When this occurs, the position of the pop-up is adjusted up and to the left so that the pop-up window will always be positioned within the bounds of the display, if possible. If the window is larger than the bounds of the display, it is positioned flush with the upper-left corner of the display.

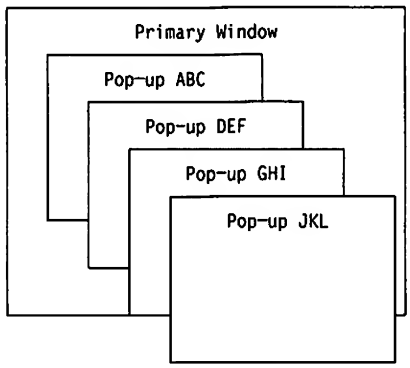


Figure 2-2. Offset Positioning for Pop-Up Windows

Chapter 3. User Interaction for Dialog Manager Applications

This chapter describes information you need to know about how the DM application will look on the screen to the user, and how the user can interact with the panel. The chapter also describes Dialog Manager validation and enter processing.

Interacting with the Panel

Once the panel has been displayed in a window, there are many ways in which the user can interact with the window or the panel. The user can move the window, and if the window is a primary window, the user can size the window. If the panel contains an action bar, the user can select an action from the action bar. The user can use the F10 key to move the cursor to the action bar, as long as F10 has not been mapped to a command other than the ACTIONS command (in the application's key mapping list).

If there are input data fields or list fields on the panel, the user can enter data or make choices from selection fields or selection lists. Either the mouse or the keyboard or both can be used. To move the cursor to the various fields on the panel, the user can use the keyboard as described below.

The user can also interact with the panel by selecting help. If provided by the application, contextual help is available on all fields on a panel, except the output-only fields. To get contextual help, the user selects help when the cursor is in the field on which help is wanted. See Chapter 9, "Using Dialog Manager Help" on page 9-1 for more information.

Keyboard Interaction

The Tab and Backtab keys are used to move the cursor among:

- Data fields (input)
- Columns in list fields (input)
- Selection fields
- Selection lists
- The command area

The Tab key moves the cursor to the next field on the panel, in the order in which the fields were defined in the tag source file. When the Tab key is used to move the cursor to the last field in the panel, the cursor "wraps" back up to the first field in the panel. The Backtab key moves the cursor in the opposite direction of the Tab key. When the Backtab key is used to move the cursor to the first field in the panel, the cursor wraps down to the last field in the panel.

Note: For fields other than list fields, pressing the Ctrl and Tab keys together functions the same as Tab, and pressing the Ctrl and Backtab keys together functions the same as Backtab.

Selection Field Keyboard Interaction

When the user uses the Tab/Backtab key to move the cursor in a selection field, the cursor is placed on the first selected choice in the selection field. If no choices are selected, the cursor is placed on the first choice. To move the cursor among the choices within a selection field, the user can use the arrow keys as follows:

- In a horizontal selection field:
  - Right/Down arrow keys move the cursor to the right.
  - Left/Up arrow keys move the cursor to the left.
- In a vertical selection field:
  - Right/Down arrow keys move the cursor down.
  - Left/Up arrow keys move the cursor up.

When the cursor reaches the end of the selection field, the arrow keys cause the cursor to wrap around to the other end of the selection field. For example, if the cursor is on the bottom choice in a vertical selection field, using the down arrow key causes the cursor to move to the first choice in the selection field.

The Tab/Backtab key must be used to move the cursor out of the selection field.

If the selection field is TYPE = ACTION and a default action is specified, the heavy border on the pushbutton moves with the cursor as the user uses the arrow keys to move among the choices. When the cursor is moved out of the selection field, the heavy border is moved back to the default choice or pushbutton.

Implicit selection is supported in single-choice selection fields, so that moving the cursor to a radio button with the arrow key automatically selects that radio button.

If mnemonics are defined within the selection field, the user can use mnemonic selection to move the cursor to the choice and select it. Mnemonic selection only works once the cursor is already placed within the selection field, because the scope of mnemonics is within one selection field only.

List Field Keyboard Interaction

The user can use several different keys to move the cursor within the list field:

Home	Moves to the first input/both column.
End	Moves to the last input/both column.
Ctrl + Home	Moves to the first row in the current column.
Ctrl + End	Moves to the last row in the current column.
Up	Moves up one row.
Down	Moves down one row.
Tab	Moves to the next input/both column. If in the last column, it moves to the first column in the next row. If in the last cell of the last column, it exits the list field and moves to the next field on the panel.
Backtab	Moves to the previous input/both column. If in the first column, it moves to the last column of the previous row. If in the first cell of the first column, it exits the list field and moves to the previous field on the panel.

Ctrl + Tab	Exits the list field and moves to the next field on the panel.
Ctrl + Backtab	Exits the list field and moves to the previous field on the panel.

Scrolling Keyboard Interaction

The PageUp, PageDown, Ctrl + PageUp, Ctrl + PageDown keys are supported to provide scrolling using the keyboard.

PageUp	Scroll to display information above that currently visible; equivalent to the DM BACKWARD command.
PageDown	Scroll to display information below that currently visible; equivalent to the DM FORWARD command.
Ctrl + PageUp	Scroll to display information to the left of that currently visible; equivalent to the DM LEFT command.
Ctrl + PageDown	Scroll to display information to the right of that currently visible; equivalent to the DM RIGHT command.

Sizing the Window

All primary windows have sizeable borders so that the user can change the size of the window.

In panels that contain a scrollable area, if the entire contents of a scrollable area are visible, neither horizontal nor vertical scroll bars appear in the window. Figure 3-1 shows the display of a panel in a window. Because the entire contents of the scrollable area are visible, no scroll bars appear in the window. The user can make the panel smaller or larger within the limits allowed by the Presentation Manager for standard windows.

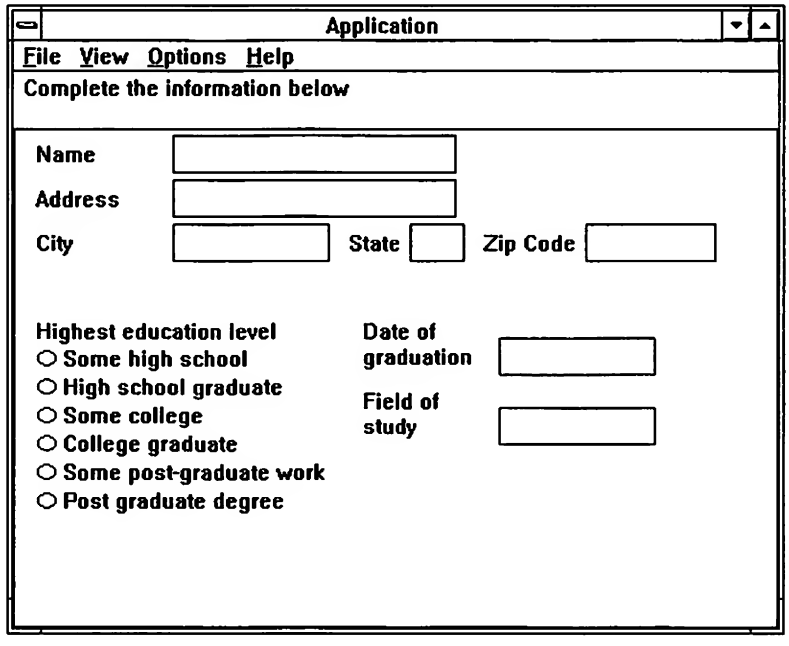


Figure 3-1. Window with All Contents Visible

Assume that the user makes the window in Figure 3-1 both horizontally and vertically smaller. The result is the panel in Figure 3-2 on page 3-4.

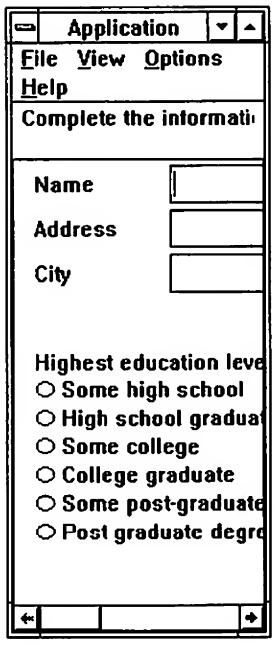


Figure 3-2. Window after Making It Smaller

The window title remains centered in the panel. The action bar choices are reformatted to fit into the new panel size, thus ensuring that the user always has access to all choices in the action bar. The scrollable panel area clips its contents, so the scroll bars become visible. This indicates to the user that more information can be seen by using the scroll bar.

In Figure 3-2, if the user were to increase the size of the panel horizontally so that the contents of the scrollable area became completely visible (horizontally), the horizontal scroll bar would disappear. The same is true for vertical sizing.

The user can make the window smaller until the minimum window size as defined by the Presentation Manager is reached.

Minimizing and Maximizing the Primary Window

The user can minimize or maximize the primary window in which a panel is displayed by selecting the **Minimize** or **Maximize** choices from the window's system menu or by selecting the window sizing icons on the title bar. When a window is minimized, it appears as an icon at the bottom of the screen. The icon specified in the ZWINICON system variable is the icon the Dialog Manager uses when the window is minimized. When a window is maximized, the window is the same size as the screen.

If the value of the ZWINICON system variable is not defined, the window changes to the size of an icon (PM default for minimized windows). If ZWINICON is defined and cannot be created by the Dialog Manager, an error is returned to the application. For more information on the ZWINICON system variable, see Chapter 16, "System Variables" on page 16-1.

Validating Fields and Panels

The Dialog Manager provides several types of validation checks that you can specify for a field.

Validating Fields

The data that a user enters is validated when the user modifies the data in a field and then uses the Tab key to move the cursor out of the field. Dialog Manager performs the validation that is defined for that field and displays an action message if the data does not pass the validation. The action message is displayed in a message pop-up that is positioned relative to the field that contains the error. When the user dismisses the message pop-up, the cursor is placed in the field that contains the error so that the user can correct the error. If the user tabs out of the field without correcting the error, the Dialog Manager does not notify the user again. All fields are validated again when the panel is submitted.

Certain types of validation, such as input-required, are not checked until the entire panel is validated.

Validating Panels

When the panel is submitted for processing, Dialog Manager validates all fields on the panel. When an error is encountered, the action message is displayed in a message pop-up relative to the field. The user must then resubmit the panel for processing. The same procedure continues until all of the errors are corrected. The validation of the fields is performed in the tabbing sequence.

The user can use the CANCEL or EXIT command to remove the panel. Dialog Manager will still validate the fields on the panel, but will not display an action message to the user. Refer to the *Dialog Tag Language Guide and Reference* for more information on validation.

Submitting the Panel for Processing

Once the user has completed interacting with the panel, the user submits the panel for processing by issuing a command that returns control to the application. Commands that return control to the application are:

- DM commands: ENTER, CANCEL, EXIT
- Application commands with a command action of PASSTHRU or SETVERB
- Commands not explicitly defined to the Dialog Manager, only if the NOMATCH = APPLCMD parameter is specified on the CMDTBL tag (refer to the *Dialog Tag Language Guide and Reference* for details).

There are two types of scenarios that can occur. Note that in all cases where Dialog Manager returns control to the application, the variable pool is updated with the values from the panel.

1. The user issues the EXIT or CANCEL command, which causes the Dialog Manager to validate the fields on the panel, but not display any action messages to the user. The validation is only performed to determine the reason code to return on the DISPLAY service call. The panel is removed and the variable pool is updated with the values on the panel.
2. The user issues the ENTER command, or any other command that returns control to the application. Dialog Manager does one of the following:

- If the panel has a command area that contains a command, the command is processed. If the command that is processed returns control to the application (and is not CANCEL or EXIT) the Dialog Manager validates the panel as described in "Validating Fields and Panels." Action messages are displayed and the variable pool is updated.
- If the command in the command area does not return control to the application (such as the EXHELP command), the command is processed, but validation does not occur and the variable pool is not updated.
- If the panel has a default action, the default action is performed. If the default action performed causes control to return to the application (and is not CANCEL or EXIT) the Dialog Manager validates the panel as described in "Validating Fields and Panels" on page 3-5. Action messages are displayed and the variable pool is updated.

  If the default action (such as PANELID) does not return control to the application, the action is performed, but validation does not occur and the variable pool is not updated.
- If the panel does not have a command area or a default action, the Dialog Manager validates the panel as described in "Validating Fields and Panels" on page 3-5. Action messages are displayed and the variable pool is updated.

Chapter 4. Calling Dialog Services

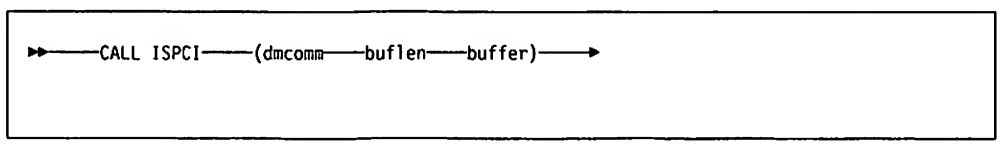
A program consists of a set of calls to the Dialog Manager plus the programming code that performs the tasks the user requests. For example, a program that computes payroll amounts might consist of calls to the Dialog Manager to display panels that request input from the user, and to get the values the user enters in the panels. In addition to the calls to the Dialog Manager, the program contains code that interprets these values and computes the payroll amounts.

When you are developing a DM application, you must decide what calls must be made to the Dialog Manager. These calls are *service calls*, and they allow you to transfer function to the Dialog Manager instead of writing the code yourself. Although the service calls provide a wide range of function, there are definitions and rules that apply to all of them. These definitions and rules provide a consistent service call interface. For language-specific requirements, see Chapter 10, "Compiling, Linking, and Running Dialog Manager Applications" on page 10-1.

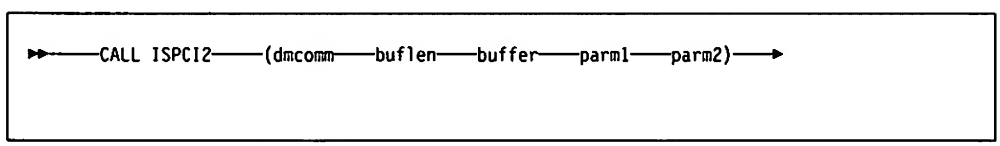
Service Call Format

The following syntax illustrates the various ways to call the dialog services in your DM application. The service call format you use depends on the services being called and the language in which your programs are written.

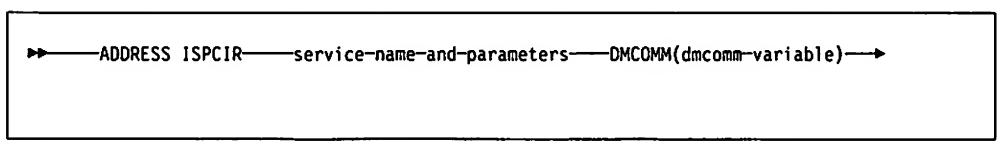
Type 1: Use the following format to call all services except VCOPY, VDEFINE, and VREPLACE.



Type 2: For the VCOPY, VDEFINE, and VREPLACE services, use the following format, which tells the Dialog Manager that two additional parameters are required.



Type 3: For applications written in Procedures Language, use the following format.



Note: Programs written in Procedures Language use a single DM entry point for all service requests, including the VCOPY service to access DM system variables.

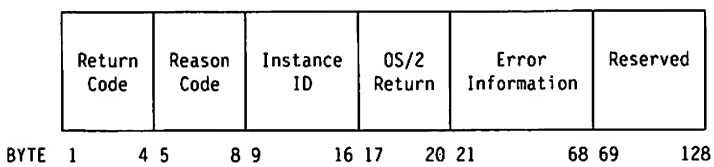
Each parameter is defined as follows:

dmcomm (the DM communication area)

An area of memory that is allocated by the DM application. The variable name associated with this area of memory is a required parameter on all the Dialog Manager service calls. The Dialog Manager uses the DM communication area to convey information, such as service call return codes, to the DM application. The entire DM communication area is reserved for the Dialog Manager use and the DM application must not modify any part of the DM communication area.

If a DM application includes multiple DMOPEN calls, the program must define a different DM communication area for each DMOPEN call. The program is responsible for pointing to the correct area for the dialog service being issued.

The size of the DM communication area is 128 bytes. The following figure shows the contents:



Byte	Contains
1 – 4	The return code from a dialog service request. The return code is a 4-byte integer value that indicates the severity of any errors returned by the Dialog Manager. Errors are indicated by the numbers 0, 4, 8, 12, and 16. If the return code value is 0, 4, or 8, the requested service call completed successfully.
5 – 8	A reason code to further define the return code. The reason code is also a 4-byte integer value.
9 – 16	The instance identifier. The Dialog Manager assigns this identifier when a dialog is established as a result of a DMOPEN service. See "Beginning and Ending the Dialog Manager Application" on page 2-1 for an explanation of how to use the instance identifier.
17 – 20	Return codes from OS/2 calls. This information can be used to determine whether a Dialog Manager problem resulted from an OS/2 call that was not valid.
21 – 68	The codes of any previous errors that may have occurred. This error trace can be used to obtain additional information about the error returned from the Dialog Manager service call.
69 – 128	Reserved.

buflen

A signed 4-byte integer containing the character length, in bytes, of the buffer.

buffer and service-name-and-parameters

The name of a program variable that contains the name of the service and its parameters. The individual service descriptions in Chapter 14, "Dialog Manager Services" specify the service name and parameters for each dialog service. The maximum buffer size before and after symbolic variable substitution, if any, is 512 bytes.

Use one or more spaces to separate buffer parameters.

If conflicting keywords are coded, the last keyword is used.

Within the buffer you can specify symbolic dialog variables. A symbolic variable is one that has an ampersand (&) as a prefix to the variable name delimited by space, a right parenthesis, or a comma (,). If the variable is undefined, it is handled like a null. Prior to a scan and syntax check of a statement, variable names and the preceding ampersands are replaced with the value of the corresponding dialog variable. A single scan takes place. For example:

```
DISPLAY PANEL(&pname)
```

The variable *pname* contains the name of the panel that is displayed.

Note: Variable substitution is not available for the DMOPEN service.

parm1, parm2
The contents of the parameters depend on the specific service call. See the specific service call description for an explanation.

Passing Dialog Variables as Parameters

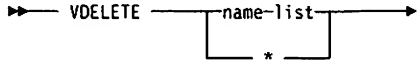
Some of the Dialog Manager services require the names of dialog variables to be passed as service parameters. These names should *not* be preceded with an ampersand unless substitution is desired. For example, suppose the buffer contains one of the following:

```
VCOPY  XYZ
VCOPY  &VNAME
```

In the first example, XYZ is the name of the dialog variable to be passed. In the second example, variable VNAME contains the name of the dialog variable to be passed.

Note: The entire buffer contents of an ISPCI call (including the service name) can be represented by a dialog variable.

Some services accept a list of variable names (*name-list*), passed as a single parameter. They are VCOPY, VDEFINE, VDELETE, and VREPLACE. For example, the syntax for the buffer in the VDELETE service is:



Name-list is a positional parameter. The number of names in the list is limited to the number of names that can be specified within the 512-byte buffer limit. Each name must be separated by a comma, one or more blanks, or a combination of a comma and blanks.

If the *name-list* consists of more than one name, it must be enclosed in parentheses. You can omit parentheses if a single name constitutes the list. For example, each of the following illustrates acceptable contents of the buffer:

```
VDELETE  (AAA,BBB,CCC)
VDELETE  (LNAME FNAME I)
VDELETE  (XYZ)
VDELETE  XYZ
```

The last two lines of the example, with and without the parentheses, are equivalent.

In other cases, a list of variable names or values can be passed in a parameter. For example, the syntax for the buffer in the VDEFINE service is:

```
VDEFINE name-list FORMAT(format) ..........
```

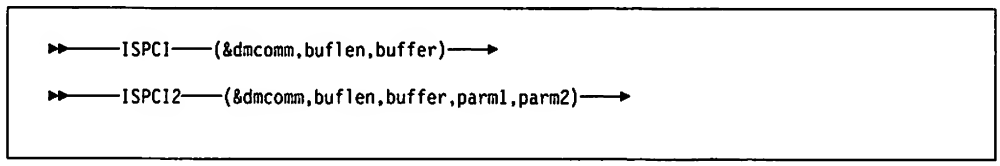
where the parentheses are required by the "keyword(value)" syntax. The value separator is the same as the one used in the *name-list*.

```
VDEFINE (var1 var2) FORMAT(CHAR FIXED) ..........
```

Calling Services in C Language

Specific Syntax

The specific call format to call dialog services in C language programs is:



Defining the DM Communication Area in C

The following code shows how to define the DM communication area in C language.

```
/* Define the error information area */

#define DMCOMMBLOCK_ERROR_INFO_COUNT 6

typedef struct
{
  unsigned long int ReasonCode;       /* Dialog Manager return code    */
  unsigned long int OS2ReturnCode     /* OS/2 return code              */
}DMERRORINFO, far *PDMERRORINFO;

/* Define the Dialog Manager communication block */

typedef struct
{
  unsigned long int ReturnCode;       /* Dialog Manager return code    */
  unsigned long int ReasonCode;       /* Dialog Manager reason code    */
  char InstanceID [8];                /* Instance identifier           */
  unsigned long int OS2ReturnCode;    /* OS/2 Return Code              */
  DMERRORINFO ErrorInfo [DMCOMMBLOCK_ERROR_INFO_COUNT];
                                      /*Error information array        */
  char filler [60];                   /* Reserved for DM--do not modify*/
}DMCOMMBLOCK, far *PDMCOMMBLOCK;
```

The above declarations can be found in the DM-provided C include file ISPCAST.H.

Usage Notes

You should ensure that the addresses, and not the values, of *parm1* and *parm2* are passed to the Dialog Manager.

The Dialog Manager provides a sample DM application written in C language. For information on locating and running this application, see "Writing Program Code to Support the Dialog" on page 1-7.

The Dialog Manager provides an include file for C language programs. This file, ISPCAST.H, provides a typedef for the DM communication area as well as procedure declarations for ISPCI and ISPCI2. To use this include file, use the following directive in your source code:

```
#include "ispcast.h"
```

C Syntax Example

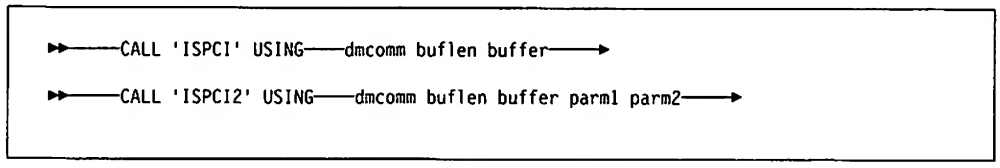
This example illustrates how to activate the Dialog Manager so that the DM application with identifier XMP1 can issue Dialog Manager service calls. Define the application variable APP_STRING, an 8-byte string, to the Dialog Manager using the name MSGNAME. Retrieve the panel named PANELX from the library contained in the application library definition file and display it. Then end Dialog Manager services.

```
#include "string.h"
#include "ispcast.h"

void main ()
{
  char buffer [50];                    /* buffer for DM service calls*/
  long int buflen;                     /* buffer length              */
  DMCOMMBLOCK dmcomm;                  /* DM communications area     */
  char app_string [8];                 /* application variable to    */
                                       /*     VDEFINE to DM          */
  long int var_length [1];             /* length array               */

  /*************************************************************************/
  /* Set up the parameters for and make a DMOPEN service call.             */
  /*************************************************************************/

  strcpy (buffer, "DMOPEN APPLID(XMP1)");
  buflen = strlen (buffer);
  ISPCI (&dmcomm, buflen, buffer);

  /*************************************************************************/
  /* Define the application variable app_string to Dialog Manager as  */
  /* the dialog variable MSGNAME.                                          */
  /*************************************************************************/

  strcpy (buffer, "VDEFINE MSGNAME FORMAT(CHAR)");
  buflen = strlen (buffer);
  var_length [0] = sizeof (app_string);
  ISPCI2 (&dmcomm, buflen, buffer, var_length, app_string);

  /*************************************************************************/
  /* Retreive and display the panel named PANELX.                          */
  /*************************************************************************/

  strcpy (buffer, "DISPLAY PANEL(PANELX)");
  buflen = strlen (buffer);
  ISPCI (&dmcomm, buflen, buffer);

  /*************************************************************************/
  /* End Dialog Manager services.                                          */
  /*************************************************************************/

  strcpy (buffer, "DMCLOSE");
  buflen = strlen (buffer);
  ISPCI (&dmcomm, buflen, buffer);
}
```

Calling Services in COBOL

Specific Syntax

The specific call format to call dialog services in COBOL programs is:



Defining the DM Communication Area in COBOL

The following code shows how to define the DM communication area in COBOL.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
*DEFINE THE DM COMMUNICATION AREA
01 DMCOMM IS COMP.
 02 DMCOMM-RETURN-CODE           PIC S9(7).
 02 DMCOMM-REASON-CODE           PIC S9(7).
 02 INSTANCE-ID                  PIC X(8).
 02 OS2-RETURN-CODE              PIC S9(7).
 02 DMERRORINFO OCCURS 6 TIMES.
  03 ERR-RETURN-CODE           PIC S9(7).
  03 ERR-OS2-RETURN-CODE       PIC S9(7).
 02 FILLER                       PIC X(60).
```

Usage Notes

Small and compact COBOL models do not support the Dialog Manager.

The Dialog Manager provides a sample DM application written in COBOL. For information on locating and running this application, see "Writing Program Code to Support the Dialog" on page 1-7.

The Dialog Manager provides an include file for COBOL language programs. The file, ISPCOBOL.INC, provides a declaration of the DM communication area. To use this include file, use the following statement in your source code:

```
COPY 'ISPCOBOL.INC'.
```

COBOL Syntax Example

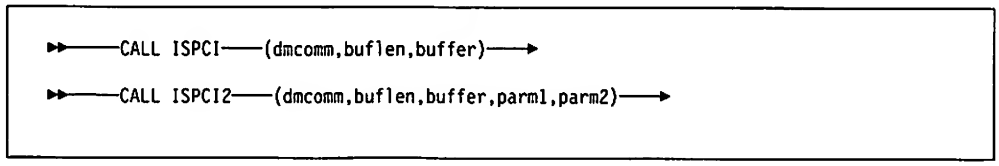
This example illustrates how to activate the Dialog Manager so that the DM application with identifier XMP1 can issue Dialog Manager service calls. Define the application variable APP_STRING, an 8-byte string, to the Dialog Manager using the name MSGNAME. Retrieve the panel named PANELX from the library contained in the application library definition file and display it. Then end Dialog Manager services.

```
*****************************************************************
*   SAMPLE COBOL DIALOG MANAGER APPLICATION                     *
*****************************************************************
 DATA DIVISION.
 WORKING-STORAGE SECTION.

*INCLUDE THE DM-PROVIDED COBOL INCLUDE FILE
 COPY 'ISPCOBOL.INC'.

*BUFFER AREA FOR DIALOG MANAGER SERVICE CALLS
 77  BUFFER              PIC X(50).
*BUFFER LENGTH
 77  BUFLEN              PIC S9(7) COMP.
*APPLICATION VARIABLE TO DEFINE TO DIALOG MANAGER
 77  APP-STRING          PIC X(8).
*LENGTH ARRAY
 01  VAR-LENGTH          OCCURS 1 TIMES.
     03 LENGTH-ENTRY     PIC S9(7) COMP.

 PROCEDURE DIVISION.

*****************************************************************
* START DIALOG MANAGER SERVICES WITH DMOPEN CALL.               *
*****************************************************************
     MOVE 'DMOPEN APPLID(XMP1)' TO BUFFER.
     MOVE 19 TO BUFLEN.
     CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

*****************************************************************
* DEFINE THE APPLICATION VARIABLE APP-STRING TO THE DIALOG      *
* MANAGER AS THE DIALOG VARIABLE MSGNAME.
*****************************************************************
     MOVE 'VDEFINE MSGNAME FORMAT(CHAR)' TO BUFFER.
     MOVE 28 TO BUFLEN.
     MOVE 8 TO LENGTH-ENTRY (1).
     CALL 'ISPCI2' USING DMCOMM BUFLEN BUFFER LENGTH-ENTRY (1)
-                APP-STRING.
*****************************************************************
* RETRIEVE THE PANEL NAMED PANELX AND DISPLAY IT.               *
*****************************************************************

     MOVE 'DISPLAY PANEL(PANELX)' TO BUFFER.
     MOVE 21 TO BUFLEN.
     CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

*****************************************************************
* END DIALOG MANAGER SERVICES.                                  *
*****************************************************************
     MOVE 'DMCLOSE' TO BUFFER.
     MOVE 7 TO BUFLEN.
     CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

     STOP RUN.
```

Calling Services in FORTRAN

Specific Syntax

The specific call format to call dialog services in FORTRAN programs is:



Defining the DM Communication Area in FORTRAN

The following code shows how to define the DM communication area in FORTRAN.

```
C   DEFINE THE DM COMMUNICATION AREA AS
C   AN ARRAY OF 32 4-BYTE INTEGERS
    INTEGER*4 DMCOMM
    DIMENSION DMCOMM (32)
    INTEGER*4 RETURNCODE
    EQUIVALENCE (DMCOMM (1), RETURNCODE)
    INTEGER*4 REASONCODE
    EQUIVALENCE (DMCOMM (2), REASONCODE)
    CHARACTER*8 INSTID
    EQUIVALENCE (DMCOMM (3), INSTID)
    INTEGER*4 OS2RETURNCODE
    EQUIVALENCE (DMCOMM (5), OS2RETURNCODE)

    INTEGER*4 ERRORINFO_REASONCODE1
    EQUIVALENCE (DMCOMM (6), ERRORINFO_REASONCODE1)
    INTEGER*4 ERRORINFO_OS2RETURNCODE1
    EQUIVALENCE (DMCOMM (7), ERRORINFO_OS2RETURNCODE1)

    INTEGER*4 ERRORINFO_REASONCODE2
    EQUIVALENCE (DMCOMM (8), ERRORINFO_REASONCODE2)
    INTEGER*4 ERRORINFO_OS2RETURNCODE2
    EQUIVALENCE (DMCOMM (9), ERRORINFO_OS2RETURNCODE2)

    INTEGER*4 ERRORINFO_REASONCODE3
    EQUIVALENCE (DMCOMM (10), ERRORINFO_REASONCODE3)
    INTEGER*4 ERRORINFO_OS2RETURNCODE3
    EQUIVALENCE (DMCOMM (11), ERRORINFO_OS2RETURNCODE3)

    INTEGER*4 ERRORINFO_REASONCODE4
    EQUIVALENCE (DMCOMM (12), ERRORINFO_REASONCODE4)
    INTEGER*4 ERRORINFO_OS2RETURNCODE4
    EQUIVALENCE (DMCOMM (13), ERRORINFO_OS2RETURNCODE4)

    INTEGER*4 ERRORINFO_REASONCODE5
    EQUIVALENCE (DMCOMM (14), ERRORINFO_REASONCODE5)
    INTEGER*4 ERRORINFO_OS2RETURNCODE5
    EQUIVALENCE (DMCOMM (15), ERRORINFO_OS2RETURNCODE5)

    INTEGER*4 ERRORINFO_REASONCODE6
    EQUIVALENCE (DMCOMM (16), ERRORINFO_REASONCODE6)
    INTEGER*4 ERRORINFO_OS2RETURNCODE6
    EQUIVALENCE (DMCOMM (17), ERRORINFO_OS2RETURNCODE6)
```

Usage Notes

The Dialog Manager provides an include file for FORTRAN language programs. This file, ISPFORT.INC, provides a declaration of the DM communication area. To use this include file, put the following compiler directive statement in your source code:

```
INCLUDE 'ISPFORT.INC'
```

In order to pass a CHARACTER*n variable as a parm2 in an ISPCI2 call, the variable must be equivalent with an integer array. For example, use the following code to define an 8-character string variable using VDEFINE:

```
INCLUDE 'ISPFORT.INC'

INTEGER         LENG*4
CHARACTER       CHARERRMSG*8
INTEGER         ERRMSG*4
DIMENSION       ERRMSG (2)
EQUIVALENCE     (ERRMSG, CHARERRMSG)

:
LENG = 8
BUFFER = 'VDEFINE ERRMSG FORMAT(CHAR)'
BUFLEN=27
CALL ISPCI2 (DMCOMM, BUFLEN, BUFFER, LENG, ERRMSG)
```

FORTRAN Syntax Example

This example illustrates how to activate the Dialog Manager so that the DM application with identifier XMP1 can issue Dialog Manager service calls. Define the application variable APP_STRING, an 8-byte string, to the Dialog Manager using the name MSGNAME. Retrieve the panel named PANELX from the library contained in the application library definition file and display it. Then end Dialog Manager services.

```
C*********************************************************************
C      DIALOG MANAGER SAMPLE FORTRAN APPLICATION
C*********************************************************************

       PROGRAM EXAMPLE1

C*********************************************************************
C      INCLUDE THE DM-PROVIDED FORTRAN INCLUDE FILE
C*********************************************************************

       INCLUDE 'ISPFORT.INC'

C      BUFFER LENGTH
       INTEGER*4 BUFLEN

C      BUFFER FOR DM SERVICE CALLS
       CHARACTER*50 BUFFER

C      APPLICATION VARIABLE AND STRING LENGTH
       CHARACTER*8 APP_STRING
       INTEGER*4 VAR_LENGTH

C      EQUIVALENCE THE CHARACTER STRING TO AN INTEGER ARRAY (FOR VDEFINE)
       INTEGER*4 ARRAY_APP_STRING
       DIMENSION ARRAY_APP_STRING (2)
       EQUIVALENCE (APP_STRING, ARRAY_APP_STRING)

C*********************************************************************
C      INITIALIZE DIALOG MANAGER.
C*********************************************************************

       BUFFER = 'DMOPEN APPLID(XMP1)'
       BUFLEN = LEN (BUFFER)
       CALL ISPCI (DMCOMM, BUFLEN, BUFFER)

C*********************************************************************
C      DEFINE THE APPLICATION VARIABLE APP-STRING TO DIALOG MANAGER
C      WITH THE NAME MSGNAME.  USE THE EQUIVALENCED INTEGER ARRAY.
C*********************************************************************

       BUFFER = 'VDEFINE MSGNAME FORMAT(CHAR)'
       BUFLEN = LEN (BUFFER)
       VAR_LENGTH = 8
       CALL ISPCI2 (DMCOMM, BUFLEN, BUFFER, VAR_LENGTH, ARRAY_APP_STRING)
```

```
C*************************************************************************
C       RETRIEVE AND DISPLAY THE PANEL NAMED PANELX.
C*************************************************************************

        BUFFER = 'DISPLAY PANEL(PANELX)'
        BUFLEN = LEN (BUFFER)
        CALL ISPCI (DMCOMM, BUFLEN, BUFFER)

C*************************************************************************
C       END DIALOG MANAGER SERVICES.
C*************************************************************************

        BUFFER = 'DMCLOSE'
        BUFLEN = LEN (BUFFER)
        CALL ISPCI (DMCOMM, BUFLEN, BUFFER)

        STOP
        END
```

Calling Services in MASM

Specific Syntax

The specific call format to call dialog services in MASM programs is:

```
ISPCI call:

push segmented address of dmcomm
push segmented address of buffer length
push segmented address of buffer
CALL ISPCI

ISPCI2 call:

push segmented address of dmcomm
push segmented address of buffer length
push segmented address of buffer
push segmented address of parm1
push segmented address of parm2
CALL ISPCI2
```

Defining the DM Communication Area in MASM

The following code shows how to define the DM communication area in Macro Assembler.

The declaration structure can be found in the Dialog Manager-provided MASM include file ISPCALL.INC.

```
;define the Dialog Manager communication block structure
dm_comm struc
dmretc              dd      ?
dmreac              dd      ?
instid              db      8 dup (?)
os2retc             dd      ?
errinfo_reac1       dd      ?
errinfo_os2retc1    dd      ?
errinfo_reac2       dd      ?
errinfo_os2retc2    dd      ?
errinfo_reac3       dd      ?
errinfo_os2retc3    dd      ?
errinfo_reac4       dd      ?
errinfo_os2retc4    dd      ?
errinfo_reac5       dd      ?
errinfo_os2retc5    dd      ?
errinfo_reac6       dd      ?
errinfo_os2retc6    dd      ?
filler              db      60 dup (?)
dm_comm ends
```

Usage Notes

Dialog Manager provides an include file for Macro Assembler language programs. This file, ISPCALL.INC, provides a declaration of the DM communication area, as well as FAR declarations of the Dialog Manager entry points. To use this include file, use the following pseudo-operation in your source code:

```
INCLUDE ISPCALL.INC
```

When you make a call to ISPCI or ISPCI2 in MASM, the Dialog Manager will automatically pop the parameters off the stack. That is, the parameters pushed onto the stack for the call to Dialog Manager will be cleared off before returning control to the DM application.

Ensure that the addresses, and not the values, of *parm1* and *parm2* are passed to the Dialog Manager.

MASM Syntax Example

This example illustrates how to activate the Dialog Manager so that the DM application with identifier XMP1 can issue Dialog Manager service calls. Define the application variable APP_STRING, an 8-byte string, to the Dialog Manager using the name MSGNAME. Retrieve the panel named PANELX from the library contained in the application library definition file and display it. Then end Dialog Manager services.

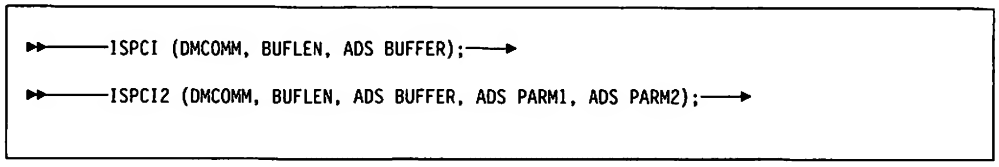
```
            TITLE EXAMPLE - masm dm example program
;*****************************************************************
            INCLUDE sysmaca.inc       ;; Include macros for OS/2 function calls

stack       SEGMENT word stack 'STACK'
            DB     1024 DUP (?)
stack       ENDS

;*****************************************************************
; DGROUP is required by OS/2 to identify the automatic data segment.
; If DGROUP is not defined in your protected mode assembler program,
; you will get the error "no automatic data segment" when you LINK
; your program.
;*****************************************************************
dgroup      group  data               ; DGROUP required by OS/2

data        SEGMENT word public 'DATA'
INCLUDE     ISPCALL.INC               ; include DM-provided masm file

dmcomm      dm_comm <>                ; DM communications area
app_string db 8 dup (' ')             ; application variable to define
var_length dd 0                       ; length variable
open_buf db  "DMOPEN APPLID(XMP1)"    ; buffer to perform DMOPEN
open_buflen dd $ - open_buf           ; length of DMOPEN buffer
vdef_buf db  "VDEFINE MSGNAME FORMAT(CHAR)"    ; buffer for VDEFINE
vdef_buflen dd $ - vdef_buf           ; length of VDEFINE buffer
disp_buf db  "DISPLAY PANEL(PANELX)"  ; buffer for DISPLAY
disp_buflen dd $ - disp_buf           ; length of DISPLAY buffer
clos_buf db  "DMCLOSE"                ; buffer to perform DMCLOSE
clos_buflen dd $ - clos_buf           ; length of DMCLOSE buffer
data        ENDS

code        SEGMENT para public 'CODE'
            ASSUME cs:code,ds:dgroup

start:      mov    ax,dgroup
            mov    ds,ax               ; Load data segment address

;*********************************************************************
; Initialize the Dialog Manager with the DMOPEN service.
;*********************************************************************
            lea    ax, dmcomm
            push   ds
            push   ax
            lea    ax, open_buflen
            push   ds
            push   ax
            lea    ax, open_buf
            push   ds
            push   ax
            call   ISPCI
```

```
;*************************************************************************
; Set the var_length variable to the size of the application string to
; define; then define the string variable to Dialog Manager using the
; name MSGNAME.
;*************************************************************************
            mov     WORD PTR var_length, length app_string
            lea     ax, dmcomm
            push    ds
            push    ax
            lea     ax, vdef_buflen
            push    ds
            push    ax
            lea     ax, vdef_buf
            push    ds
            push    ax
            lea     ax, var_length
            push    ds
            push    ax
            lea     ax, app_string
            push    ds
            push    ax
            call    ISPCI2

;*************************************************************************
; Retrieve and display the panel named PANELX.
;*************************************************************************
            lea     ax, dmcomm
            push    ds
            push    ax
            lea     ax, disp_buflen
            push    ds
            push    ax
            lea     ax, disp_buf
            push    ds
            push    ax
            call    ISPCI

;*************************************************************************
; End Dialog Manager services with DMCLOSE.
;*************************************************************************
            lea     ax, dmcomm
            push    ds
            push    ax
            lea     ax, clos_buflen
            push    ds
            push    ax
            lea     ax, clos_buf
            push    ds
            push    ax
            call    ISPCI

quit:       @DosExit        1,0         ;; Exit with return code of 0

code        ENDS
            END     start
```

Calling Services in Pascal

Specific Syntax

The specific call format to call dialog services in Pascal programs is:



Defining the DM Communication Area in Pascal

The following code shows how to define the DM communication area in Pascal.

```
{DEFINE THE ERROR INFORMATION ARRAY}
TYPE
 ERRINF_STRUCT=RECORD
  REA_CODE: INTEGER4;
  OS2_RETC: INTEGER4;
END;

{DEFINE THE DM COMMUNICATION AREA}
TYPE
 COMBLOCK=RECORD
  RET_CODE:                 INTEGER4;
  REA_CODE:                 INTEGER4;
  INSTID:                   STRING (8);
  OS2_RETC:                 INTEGER4;
  ERRINFO:                  ARRAY[1..6] OF ERRINF_STRUCT;
  FILLER:                   STRING (60);
END;
```

Usage Notes

Dialog Manager provides an include file for Pascal language programs. This program, ISPPASC.INP, provides a declaration of the DM communication area, as well as procedure definitions of the Dialog Manager procedures. To use this include file, use the following metacommand in your source code:

```
(*$INCLUDE 'ISPPASC.INP'*)
```

The first two parameters to the ISPCI and ISPCI2 functions must be declared in the procedure declaration with VARS.

If BUFFER is defined in an application as STRING(*n*), then the calls to ISPCI and ISPCI2 must be in the following form:

```
ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
ISPCI2 (DMCOMM, BUFLEN, ADS BUFFER, ADS parm1, ADS parm2);
```

If BUFFER is defined in an application as LSTRING(n), then the calls to ISPCI and ISPCI2 must be in the following form:

```
ISPCI (DMCOMM, BUFLEN, ADS BUFFER [1]);
ISPCI2 (DMCOMM, BUFLEN, ADS BUFFER [1], ADS parm1, ADS parm2);
```

To use a variable defined as LSTRING(*n*) as parm2 in an ISPCI2 call, the syntax must be in the following form:

```
ISPCI (. . ., ADS parm2 [1]);
```

The application developer should ensure that the addresses, and not the values, of *parm1* and *parm2* are passed to Dialog Manager.

Pascal Syntax Example

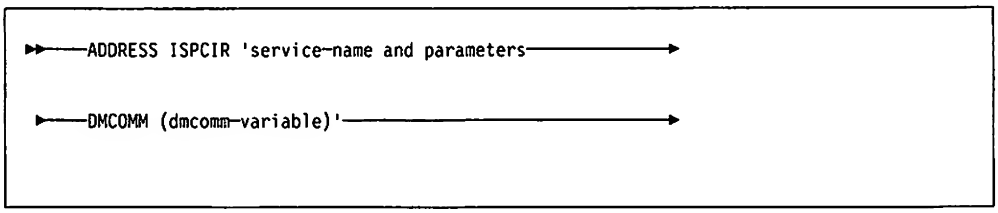
This example illustrates how to activate the Dialog Manager so that the DM application with identifier XMP1 can issue Dialog Manager service calls. Define the application variable APP_STRING, an 8-byte string, to the Dialog Manager using the name MSGNAME. Retrieve the panel named PANELX from the library contained in the application library definition file and display it. Then end Dialog Manager services.

```
(*********************************************************************)
(*              SAMPLE DIALOG MANAGER PASCAL APPLICATION             *)
(*********************************************************************)
PROGRAM EXAMPLE;

(* INCLUDE THE DM-PROVIDED PASCAL INCLUDE FILE *)
(*$INCLUDE: 'ISPPASC.INP'*)

VAR [PUBLIC]
  DMCOMM:COMBLOCK;                    (* DM COMMUNICATIONS AREA     *)
  BUFFER:STRING(50);                  (* BUFFER FOR DM SERVICE CALLS*)
  BUFLEN:INTEGER4;                    (* BUFFER-LENGTH VARIABLE     *)
  APP_STRING:STRING(8);               (* APP VARIABLE TO VDEFINE    *)
  VAR_LENGTH:INTEGER4;                (* LENGTH VARIABLE FOR VDEFINE*)

BEGIN { EXAMPLE }

(*********************************************************************)
(* INITIALIZE THE DIALOG MANAGER WITH THE DMOPEN SERVICE.            *)
(*********************************************************************)

   COPYSTR ('DMOPEN APPLID(XMP1)', BUFFER);
   BUFLEN := 19;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

(*********************************************************************)
(* DEFINE THE APPLICATION VARIABLE APP_STRING TO DIALOG MANAGER USING *)
(* THE NAME MSGNAME.                                                 *)
(*********************************************************************)

   COPYSTR ('VDEFINE MSGNAME FORMAT(CHAR)', BUFFER);
   BUFLEN := 28;
   VAR_LENGTH := 8;
   ISPCI2 (DMCOMM, BUFLEN, ADS BUFFER, ADS VAR_LENGTH, ADS APP_STRING);

(*********************************************************************)
(* RETRIEVE AND DISPLAY THE PANEL NAMED PANELX.                      *)
(*********************************************************************)

   COPYSTR ('DISPLAY PANEL(PANELX)', BUFFER);
   BUFLEN := 21;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

(*********************************************************************)
(* END DIALOG MANAGER SERVICES.                                      *)
(*********************************************************************)

   COPYSTR ('DMCLOSE', BUFFER);
   BUFLEN := 7;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

END. { EXAMPLE }
```

Calling Services in Procedures Language

Specific Syntax

The specific call format to call dialog services in Procedures Language programs is:



Note: The service name and the parameters are those that would appear in the buffer for a service call to the ISPCI entry point.

Defining the DM Communication Area in Procedures Language

The following shows how Dialog Manager implicitly defines the DM communication area in the stem variable specified on the DMCOMM parameter.

```
dmcomm-variable.1     - return code
dmcomm-variable.2     - reason code
dmcomm-variable.3     - instance ID
dmcomm-variable.4     - OS/2 return code
dmcomm-variable.5-16  - error information
```

Usage Notes

The following services are not supported: VDEFINE, VDELETE, VREPLACE, and VRESET.

In Procedures Language, a DM application is considered to be a Presentation Manager program. Therefore, a DM application needs to run in a Presentation Manager window. To enable your DM application to run in a windowing environment, you need to initiate the application with the following statement:

```
PMREXX programname
```

where *programname* is the name of your application.

Procedures Language Subcommand Interface: To call a Dialog Manager service, use the Procedures Language subcommand facility rather than the CALL interface. The ISPCI program call interface currently defined is modified to pass *dmcomm* as a parameter. The Dialog Manager subcommand processor must be registered by the application prior to issuing a DM service request and prior to issuing the first ISPCIR command. The subcommand processor registration is required because the Dialog Manager is a set of callable services and does not establish an environment at initialization.

To register the Dialog Manager subcommand processor, use the following Procedures Language statement:

```
RXSUBCOM REGISTER ISPCIR ISPCIR ISPCIR
```

After your DM application has completed processing, issue the following Procedures Language statement to restore the environment to its previous state:

```
RXSUBCOM DROP ISPCIR
```

Procedures Language Subcommand Environment: Dialog Manager services are called using the ISPCIR command. The general format for issuing a command is:

```
ADDRESS ISPCIR service-name parameter1 parameter2 ...
```

To eliminate the need to specify ADDRESS and ISPCIR on each Dialog Manager service request, include the following Procedures Language statement prior to issuing the first ISPCIR command:

```
ADDRESS ISPCIR
:
service-name parameter-list
```

The service name and parameters serve the same purpose as the buffer passed on the ISPCI call interface. DMCOMM is passed as another parameter at the end. The dialog passes a stem name to the Dialog Manager as the parameter value. The following example illustrates the syntax for calling the DISPLAY service:

```
DISPLAY PANEL(panel-name) MSG(messag-id) CURSOR(field-name)
        CSRINDEX(cursor-index) CSRPOS(cursor-position)
        DMCOMM(dmcomm-variable)
```

The *dmcomm-variable* is a Procedures Language stem variable name that Dialog Manager uses to set the following Procedures Language variables:

dmcomm-variable.1
The return code. In addition, the Procedures Language special variable, RC, is set with the return code.

dmcomm-variable.2
The reason code.

dmcomm-variable.3
The instance ID, which is set at DMOPEN time only.

dmcomm-variable.4
The OS/2 return code.

dmcomm-variable.5-16
Error information. These variables are set only when Dialog Manager detects an error. These variables contain the OS/2 and Dialog Manager return codes for the last six errors detected by the Dialog Manager.

The dialog must avoid using any variable that starts with the same stem. The Dialog Manager could use them for information relative to this dialog.

If the *dmcomm* parameter is not specified, the *dmcomm-variable* defaults to DMCOMM. Do not use the default if there are multiple DMOPENs because each DMOPEN requires a unique DM communication area.

Note: When sharing a dialog among multiple DMOPENs, the *instance-id* coded on the INSTID keyword can be specified by placing the appropriate *dmcomm-variable.3* outside of quotes. For example,

```
ADDRESS ISPCIR 'DMOPEN APPLID(ABC) INSTID(' DM1.3 ') DMCOMM(DM2)'
```

Any subsequent Dialog Manager service request relating to a particular DMOPEN requires that the *dmcomm-variable* stem name be passed on the command string so that the Dialog Manager can tell which DMOPEN this request targets.

The purpose of the *dmcomm* stem variable in the Procedures Language interface and in the *dmcomm* parameter on the ISPCI call interface is identical.

The Procedures Language interpreter calculates the buffer length (BUFLEN) automatically and passes it to the subcommand processor.

The following example illustrates the use of the *dmcomm* keyword:

```
/*****************************************************************/
/* Set up subcommand address.                                    */
/*****************************************************************/
ADDRESS ISPCIR
:
/*****************************************************************/
/* Open a dialog.                                                */
/*****************************************************************/
'DMOPEN APPLID(ABC) DMCOMM(A1)'
/*****************************************************************/
/* DM will set the following upon return:                        */
/*    A1.1     = return code                                     */
/*    A1.2     = reason code                                     */
/*    A1.3     = instance ID                                     */
/*    A1.4     = OS/2 return code                                */
/*    A1.5-16  = error information, if any                       */
/*****************************************************************/

/*****************************************************************/
/* Display a panel.                                              */
/*****************************************************************/
'DISPLAY PANEL(PAN1) DMCOMM(A1)'
/*****************************************************************/
/* DM will set the following upon return:                        */
/*    A1.1     = return code                                     */
/*    A1.2     = reason code                                     */
/*    A1.3     = instance ID                                     */
/*    A1.4     = OS/2 return code                                */
/*    A1.5-16  = error information, if any                       */
/*****************************************************************/
```

Procedures Language Syntax Example

This example illustrates how to register the Dialog Manager subcommand processor and establish the Dialog Manager subcommand environment. Next, activate the Dialog Manager so that the DM application with identifier XMP1 can issue Dialog Manager service calls. Retrieve the panel named PANELX from the library contained in the application library definition file and display it. Then end Dialog Manager services and drop the subcommand environment.

```
/*********************************************************************/
/* Procedures Language Sample Dialog Manager Application             */
/*********************************************************************/

/*********************************************************************/
/* Register Dialog Manager subcommand processor.                     */
/*********************************************************************/

RXSUBCOM REGISTER ISPCIR ISPCIR ISPCIR

/*********************************************************************/
/* Initialize Dialog Manager.                                        */
/*********************************************************************/

ADDRESS ISPCIR 'DMOPEN APPLID(XMP1) DMCOMM(DM1)'

/*********************************************************************/
/* VDEFINE is not supported for Procedures Language.                 */
/*********************************************************************/

/*********************************************************************/
/* Retrieve and display the panel named PANELX.                      */
/*********************************************************************/

ADDRESS ISPCIR 'DISPLAY PANEL(PANELX) DMCOMM(DM1)'

/*********************************************************************/
/* End Dialog Manager.                                               */
/*********************************************************************/

ADDRESS ISPCIR 'DMCLOSE DMCOMM(DM1)'

/*********************************************************************/
/* Drop the subcommand environment.                                  */
/*********************************************************************/

RXSUBCOM DROP ISPCIR
```

Chapter 5. Using Message Facilities

At this point in developing your DM application, you can add the support of the Dialog Manager message facilities. Dialog Manager message facilities allow communication from the DM application or the Dialog Manager to the user in the form of a message. When the message text is retrieved, variable substitution within the message text is performed if necessary. The maximum length a message can have, after variable substitution is performed, is 512 bytes. The DM application can request the display of a message by specifying the MSG parameter on the DISPLAY service call. See Chapter 14, "Dialog Manager Services" on page 14-1 for a detailed description of this service call.

Message Types

All Dialog Manager messages displayed by the Dialog Manager must be defined using the MSG and MSGMBR tags. Refer to the *Dialog Tag Language Guide and Reference* for a complete description of these tags.

You can use the MSG tag and its MSGTYPE attribute to define three types of messages:

Information Use information messages to convey to the user that a program is performing normally or has performed normally. For example, an information message could tell the user that a requested task, such as a database search, is complete.

Warning Use warning messages to tell the user that a potentially undesirable situation could occur. For example, a warning message could tell the user that the disk is almost full. Unless the user overrides it, the alarm sounds when the message is displayed.

Action Use action messages to notify the user that an exception condition has occurred. Action messages require the user to take action related to the exception condition. Unless the user overrides it, the alarm sounds when the message is displayed.

Displaying a Message in a Dialog Manager-Created Pop-Up Window

All Dialog Manager-created pop-ups used to display messages have these characteristics.

- The user cannot interact with any underlying primary or pop-up window while the pop-up containing the message is on the screen. The cursor is placed in the pop-up containing the message.
- The pop-up window cannot be sized.
- The size of the pop-up window is determined by the Dialog Manager according to the length of the message. Dialog Manager ensures that all of the message text is visible, thus the pop-up window is not scrollable.
- Dialog Manager sounds an alarm when displaying a warning or action message.
  - The message pop-up buttons are:
    - OK – Removes the pop-up window.

— Help – Displays the help panel for the specific message. The user can interact with both the help panel and the pop-up window in which the message appears.

Positioning for Message Pop-Ups Created by Dialog Manager

The DM application can request that the message pop-up created by the Dialog Manager be positioned relative to a particular field in the displayed panel. You do this by using the MSGLOC parameter when the MSG parameter is specified on the DISPLAY service call.

Specifying the MSGLOC parameter on the service call tells the Dialog Manager to position the message pop-up in relation to the specified field on the current panel when the message is displayed.

Pop-up windows positioned relative to a field are positioned according to the Dialog Manager field-adjacent positioning rules. See "Field-Adjacent Positioning" on page 2-6 for more information.

Displaying the Message ID

Every message defined using the MSG and MSGMBR tags has a unique *message-id*. The *message-id* consists of the message member name (specified on the MSGMBR tag) concatenated with the message suffix (specified by the SUFFIX attribute on the MSG tag). The user can choose to have the message ID displayed or not displayed as part of the message text. The user makes this choice using the Dialog Manager CHGDEFS command and the choice is reflected in the value of the system variable ZMSGID.

Application Displayed Message Panels

The DM application also has the option of displaying messages in pop-up windows. This would be desirable if the DM application wanted to include entry fields or selection fields on the panel with the message text. The ADDPOP and DISPLAY service calls give the DM application this ability. To display a message in a pop-up window, the DM application should:

- **At panel definition time:**
  1. Use the PANEL tag to define the message panel, any fields that should be on the message panel, and the message type. If the message is a warning message or an action message, specifying the MSGTYPE parameter on the panel definition sounds the alarm.
  2. Use the INFO tag to create the message text.
- **At run time:**
  1. Call the ADDPOP service so that the subsequent DISPLAY call will display the panel in a pop-up window. If the pop-up message panel should be positioned relative to a field, then specify the name of that field using the POPLOC parameter on the ADDPOP service call.
  2. Use the DISPLAY service call to display the panel. If the message is a warning message or an action message, the alarm will sound.

Positioning of Application-Displayed Message Pop-Ups

When a DM application displays its own message pop-ups, it can position the pop-up window relative to a field. This is done by using the POPLOC parameter on the ADDPOP service call. If there is more than one field on the panel with the specified field name, the first one is used. When positioning pop-up windows relative to a field, the Dialog Manager uses field-adjacent positioning rules. See "Field-Adjacent Positioning" on page 2-6 for more information. If the POPLOC parameter is not specified on the ADDPOP service call, the Dialog Manager positions the pop-up window using offset positioning rules. See "Offset Positioning" on page 2-6 and "Using Pop-up Windows" on page 2-3 for more information.

The following example illustrates how the ADDPOP POPLOC parameter and MSG MSGLOC parameters on DISPLAY work together:

```
DISPLAY  PANEL(PANELA)
  ADDPOP POPLOC(FIELDX)
  DISPLAY PANEL(PANELB) MSG(MSGA000) MSGLOC(FIELDY)
```

This results in Figure 5-1.

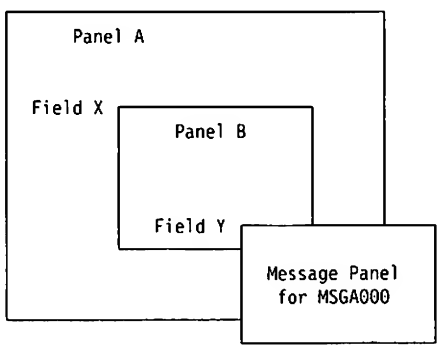


Figure 5-1. Field-Adjacent Message Pop-Up Positioning

Displaying Confirmation Messages

The LOCK parameter on the DISPLAY service can be used to display confirmation messages and immediately return control to the application once the user submits the message pop-up for processing.

Suppose you have the following code that displays a panel TRANS, a message pop-up CONFA001, and another panel MAINPAN.

```
DISPLAY PANEL(TRANS)
/*  process user's request  */
DISPLAY MSG(CONFA001)
DISPLAY PANEL(MAINPAN)
```

Once the user submits the TRANS panel, control returns to the application and the user's request is processed. Then the message pop-up containing the message CONFA001 is displayed to confirm that the request has been completed. The user must submit the message pop-up, CONFA001, and then the TRANS panel before control returns to the application so that the panel MAINPAN can be displayed.

By adding the LOCK parameter to the DISPLAY service call, you can by-pass the process described above. For example:

```
DISPLAY PANEL(trans)
/*  process user's request  */
DISPLAY MSG(confa001) LOCK
DISPLAY PANEL(mainpan)
```

In this example, the LOCK parameter has been added to the DISPLAY call. Once the user submits the TRANS panel, control returns to the application and the user's request can be processed. Then the message pop-up containing the message CONFA001 is displayed to confirm that the request has been completed. Once the user submits the message pop-up, control returns immediately to the application, and the MAINPAN panel is displayed.

Chapter 6. Using Command Facilities

Commands are one of the fundamental building blocks of the Dialog Manager. Commands can be:

- Associated with action bar pull-down choices
- Associated with choices in selection fields
- Assigned to keys
- Typed in a command entry field.

In any of these cases, the Dialog Manager processes the command in the same way. As a result, if the DM application handles the request, its interface to Dialog Manager will be the same, irrespective of how the user executes the command.

Refer to the descriptions of the PDC, ACTION, CHOICE, CMDAREA, and KEYI tags in the *Dialog Tag Language Guide and Reference* for more information on defining commands in your DM application.

The Dialog Manager supports a number of built-in commands. These DM-provided commands are described in Chapter 15, "Dialog Manager Commands and Keys" on page 15-1. In addition to using these commands, a DM application can:

- Define private commands that are not explicitly defined to the Dialog Manager.
- Provide alternate support for the Dialog Manager commands.
- Define additional commands to the Dialog Manager. See "Command Processing Support" on page 6-2 for information about the support the Dialog Manager provides for these commands.

In the preceding cases, the DM application must define an application-level command table. (Refer to the *Dialog Tag Language Guide and Reference* for more information on defining the command table.) Using a command table, the DM application can specify:

- Whether unrecognized commands are to be passed to the DM application.

  Commands entered by the user that are not recognized by the Dialog Manager (either as application-level commands or built-in commands) can be passed to the DM application in the system variable ZCMD. See "System Variables" on page 7-6 for more information. This action occurs only when an application-level command table is active and that application-level command table was created with the NOMATCH=APPLCMD attribute on the CMDTBL tag.

  If the application-level command table was defined with the NOMATCH=ERROR attribute on the CMDTBL tag or if no application-level command table is active, the Dialog Manager will display an action message for unrecognized commands. In addition, the Dialog Manager displays action messages for commands issued by the user that are either not valid or not available.

- Alternate handling of Dialog Manager commands:
  - Make a command unavailable
  - Pass a command to the DM application
  - Give a command a different definition.

- The internal and external name definitions of commands

- What action should be taken when the command is issued.

Command Processing Support

The Dialog Manager command processing component provides the following support for application-defined and Dialog Manager-provided commands:

- Command help (help for a specific command). Refer to the *Dialog Tag Language Guide and Reference* for more information.
- Command retrieval. See Chapter 15, "Dialog Manager Commands and Keys" on page 15-1 for more information about the RETRIEVE command.

Defining the Command Action

Each command that is defined in a command table has an action associated with it. This command action is specified by the CMDACT tag. The valid actions are:

CLASS The CLASS action allows DM applications to pass control to an application user exit. See "User Controls and User Exits" on page 12-8 for more information on user exits.

ALIAS The ALIAS action executes another internal command. The command to be executed must exist after the current command in the command table search.

PASSTHRU The PASSTHRU action returns control to the DM application from the DISPLAY service. The command name (*internal-command-name*) and command parameters (if any) are returned to the DM application in the system variable ZCMD. ZCMD is updated in the dialog variable pool with this value. For example:

```
<cmd name=print>Print
  <cmdact action=passthru>
```

In this example, the dialog has passed the Print command. During Dialog Manager processing, if you enter Print, the verb *PRINT* is stored in the system variable ZCMD and control returns to the DM application, which determines the action to be taken.

SETVERB In addition to the PASSTHRU action, you can also pass commands to the DM application using the SETVERB action. This action returns control to the DM application from the DISPLAY service. The command name (*internal-command-name*) is returned to the DM application in the system variable ZVERB, and the left-justified command parameters (if any) are returned to the DM application in the system variable ZCMD. Both ZVERB and ZCMD are updated in the dialog variable pool with these values. For example:

Query Dept T54

```
<cmd name=query>Query
  <cmdact action=setverb>
```

In this example, the dialog has defined the Query command. During Dialog Manager processing, if the user enters Query Dept T54 in a valid command entry field, *QUERY* is stored in the system variable ZVERB and the string "Dept T54" is stored in system variable ZCMD. Control returns to the DM application, which determines the action to be taken.

In addition to the command actions already described, the DM-provided commands listed below can be specified as the command action for any command. See Chapter 15, "Dialog Manager Commands and Keys" on page 15-1 for more information.

BACKWARD	HELP
CANCEL	HELPHELP
CHGDEFS	INDEX
ENTER	KEYS
EXIT	LEFT
EXHELP	PANELID
FKA	RETRIEVE
FORWARD	RIGHT

Note: The system variables ZVERB and ZCMD will not be available (created) in the dialog variable pool until either the application defines, updates, or copies some value into them or until one of the following Dialog Manager actions occurs:

- ZVERB is updated in the dialog variable pool when:
  - The CANCEL action is run. CANCEL is stored in the system variable ZVERB.
  - The EXIT action is run. EXIT is stored in the system variable ZVERB.
  - An application command is run, and the command has been assigned the action SETVERB.

  Only the *internal-command-name* is stored in the ZVERB system variable.
- ZCMD is updated in the dialog variable pool when:
  - Any of the preceding Dialog Manager actions occur. Only command parameters are stored in the system variable ZCMD.
  - An application command is executed, and the command has been assigned the action PASSTHRU.

    The *internal-command-name* and command parameters (if any) are stored in the ZCMD system variable.
  - A command cannot be found in the search sequence, and the application command table that is currently active was created with the NOMATCH=APPLCMD attribute.

    The *internal-command-name* and command parameters (if any) are stored in the ZCMD system variable.

If the user issues a VCOPY to copy either the ZVERB or ZCMD system variables and these variables have not been created, the Dialog Manager returns a warning condition.

Refer to the CMDACT tag in the *Dialog Tag Language Guide and Reference* for additional information and for examples of how to use the attributes of the CMDACT tag.

Specifying Command Actions Dynamically: You can use a dialog variable to specify a command action dynamically. Suppose, for example, an application command table includes the following entries:

```
<cmd name=scroll>SCROLL
  <cmdact action='%SCROLLACT'>
```

You can use the variable SCROLLACT to dynamically control the action of the SCROLL command as follows:

- If SCROLLACT is set to PASSTHRU, the SCROLL command is returned to the DM application in the system variable ZCMD.
- If SCROLLACT is set to BACKWARD and the BACKWARD command has not been overridden by an application command, the action associated with the BACKWARD command is performed when the SCROLL command is issued.
- If SCROLLACT is set to ALIAS FORWARD, command scanning continues to find the FORWARD command. If the FORWARD command is found, the action associated with the FORWARD command is performed.

If you specify an invalid action in the variable, the Dialog Manager returns an error condition.

Chapter 7. Using Dialog Variables, the Dialog Variable Pool, and Variable Services

Dialog variables are a means by which your DM application communicates with the Dialog Manager. This chapter describes how the Dialog Manager handles the variables in your DM application, the services you use to manipulate dialog variables, and the rules to follow when referencing variables in the Dialog Manager services. In addition, this chapter introduces the system variables that can be included in a DM application.

Dialog Variable Pool

Dialog variables are organized into a group called a *dialog variable pool*. When the Dialog Manager encounters a dialog variable name in a service, panel, or message, it searches the pool to access the dialog variable's value.

A dialog variable pool is created by Dialog Manager when a DMOPEN service call is made. A new variable pool is created for each DMOPEN request.

An application must supply a DM communication area (*dmcomm*) on each DMOPEN call. The Dialog Manager uses the DM communication area to reference the pool established by the DMOPEN call. As a result, when an application issues a Dialog Manager service, the *dmcomm* value specified on the service call determines the dialog variable pool to be used. The Dialog Manager provides no facility to share dialog variables across DMOPEN service calls.

Note: For Procedures Language applications, the Dialog Manager does not create a dialog variable pool. Instead, it uses the Procedures Language variable pool as the dialog variable pool. Therefore, multiple DMOPENs issued by a single Procedures Language dialog will not result in the creation of unique dialog variable pools.

Types of Variables

A dialog variable pool can contain two types of variables:

- Explicitly defined dialog variables.

  The VDEFINE service is used to explicitly define the location (by address) of dialog variables. The addresses are used by the Dialog Manager to store and retrieve data directly from application owned storage. This allows a DM application to declare a program variable, define it to the Dialog Manager, issue a DISPLAY call for a panel containing the variable, and perform normal operations such as "IF VAR1 = 3 THEN..." when control returns to the application.

  The Dialog Manager assumes that no other programs will attempt to update the storage for explicitly defined dialog variables while a Dialog Manager service is processing.

  If a DM application uses a second (or later) VDEFINE call before a corresponding VDELETE call with the same variable name, the Dialog Manager:

  - Suspends the current definition (pushes it down in a stack)
  - Makes the latest definition current (places it on top of the stack).

A VDELETE call with this variable name causes the Dialog Manager to remove the current definition from the top of the stack, and the previous definition for that variable becomes current. This facility is known as *stacked* VDEFINE.

The stacked VDEFINE facility is especially useful when DM application subroutines use the VDEFINE service for the same variable name, but with different characteristics for it, such as storage location and data type. Because the Dialog Manager does not recognize a DM application subroutine, it does not establish a new pool, and must use the VDEFINE stack to keep track of the variables. The DM application subroutine should issue a VDELETE call for all dialog variables that it defines using the VDEFINE service before returning to the calling code. The Dialog Manager uses the storage for the variable as it was last defined (unless it is deleted).

- Implicitly defined dialog variables.

  When the Dialog Manager encounters a dialog variable name in a service, panel, message or command table that has not been explicitly defined by the DM application, it implicitly defines a dialog variable. The Dialog Manager initially gives implicitly defined variables a null value, but they may take on other values as the result of dialog processing such as input into a panel field during DISPLAY. Also, the DM application can use the VCOPY and VREPLACE services to directly access the value of an implicit variable.

  An explicitly defined dialog variable and an implicitly defined dialog variable can have the same name. This can occur when, using the VDEFINE service, a DM application creates a defined variable using the same name as an existing implicit variable. When this happens, only the explicitly defined value can be accessed. To make the implicit entry accessible, the DM application must use the VDELETE service to remove any defined entries for that variable name made through the VDEFINE service.

  Notes:

  1. VDELETE calls do not affect implicitly defined dialog variables.
  2. All implicitly defined variables are character format. Arrays must be explicitly defined using the VDEFINE service.

  The Dialog Manager also contains a set of system variables that provide useful information to the DM application. See Chapter 16, "System Variables" on page 16-1 for further information.

When the DISPLAY service returns control to the Dialog Manager, all variables associated with the displayed panel are updated in the dialog variable pool. This includes any variables changed as a result of the user's making selections on the panel or entering data into data fields. The application can ignore this data if the user cancels the operation after making some changes. To do this, the application must either use the dialog variable pool as a staging area for data, using the data only if the user submits the panel for processing, or restore all the data in the pool that could have been changed by the panel if the user cancels the operation.

Variable Services

The variable services provided by the Dialog Manager and a brief description of each follows. For a complete functional description of each service and its syntax, see Chapter 14, "Dialog Manager Services" on page 14-1.

VCOPY

The VCOPY service allows an application to obtain a copy of a dialog variable. Since explicitly defined dialog variables exist in application storage, the VCOPY service would typically be used to access implicitly defined dialog variables and Dialog Manager system variables.

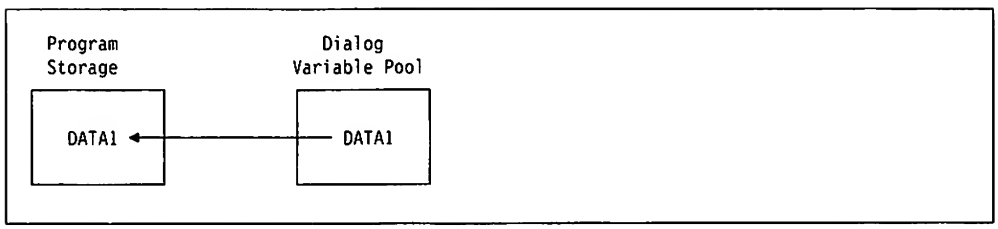


Figure 7-1. VCOPY Service

VDEFINE

The VDEFINE service associates a dialog variable name (for example, using the DATAVAR attribute on the DTAFLD tag) with program storage. The variables are considered to be explicitly defined dialog variables. Explicitly defined variables are automatically available to either the Dialog Manager or the DM application after either updates them. You can also define a list of variables with a single service call.

If a variable of the same name already exists in the dialog variable pool, the existing definition is pushed down and the new definition becomes the currently active definition. This is called the stacked VDEFINE facility.

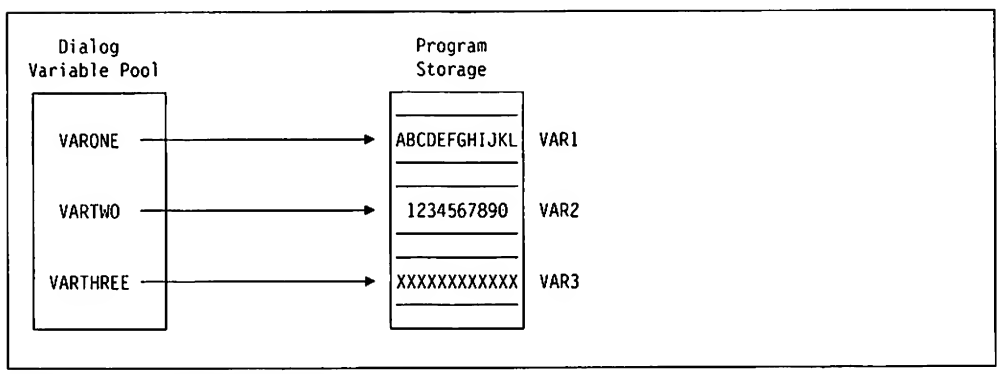


Figure 7-2. VDEFINE Service

VDELETE

The VDELETE service allows an application to delete a dialog variable from the dialog variable pool. The variable must have been previously defined using the VDEFINE service call. If more than one definition of a variable name exists in a VDEFINE stack, VDELETE will remove the top definition on the stack and the next highest one will become current.

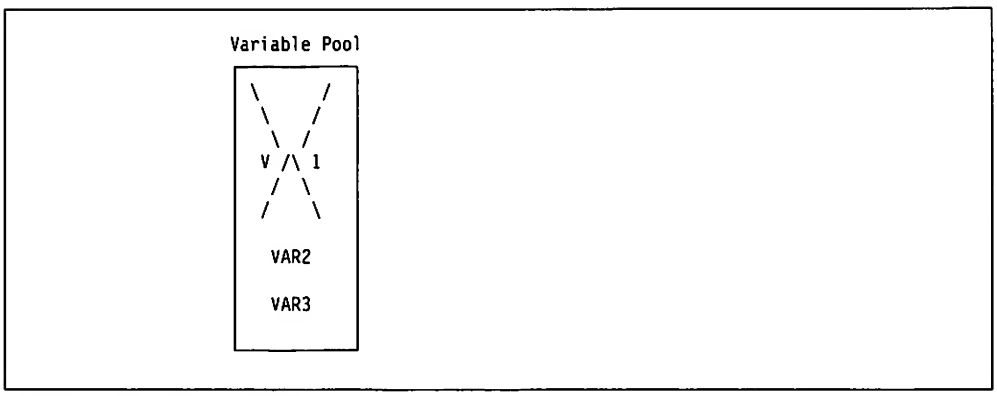


Figure 7-3. VDELETE Service

VREPLACE

The VREPLACE service allows a DM application to update the value of a dialog variable in the dialog variable pool. If the variable does not exist in the dialog variable pool, it will be created as an implicitly defined dialog variable. Multiple variables can be updated with one service call by specifying a name list.

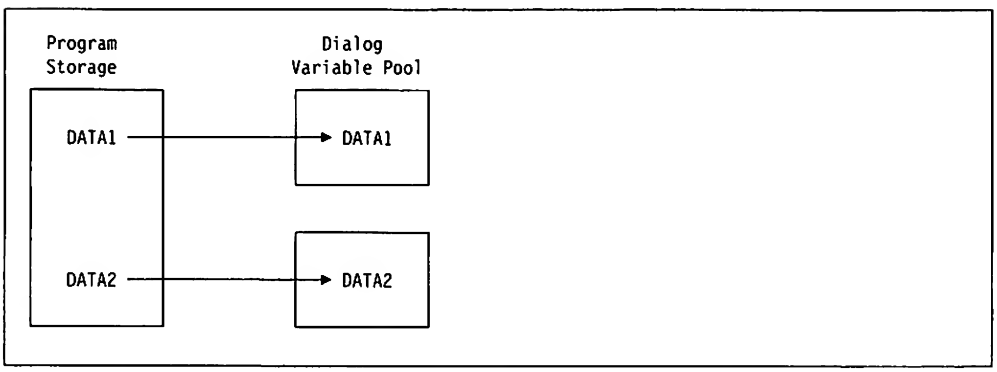


Figure 7-4. VREPLACE Service

VRESET

The VRESET service allows a DM application to reset its dialog variable pool to empty. Any reference to dialog variables after the VRESET service implicitly defines these variables.

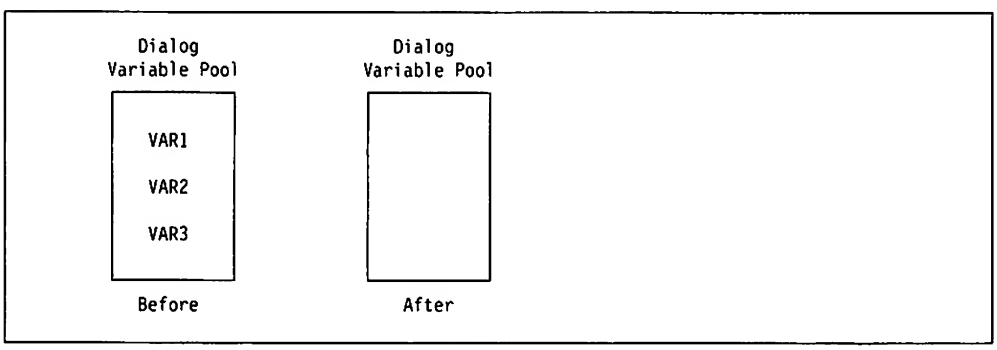


Figure 7-5. VRESET Service

Note: The VDEFINE, VDELETE, VREPLACE, and VRESET services are not supported for Procedures Language dialogs. The VCOPY service is used in Procedures Language dialogs to access Dialog Manager system variables.

Dialog Variable Names and Lengths

Dialog variables are referenced by using their names in all of the variable service calls. This section describes rules for naming dialog variables.

Dialog variable names must conform to the following rules:

- 1 – 8 characters in length
- 1st character: a – z, A – Z

  Note: All Dialog Manager system variables begin with the letter Z. Do not define your own non-system dialog variables beginning with the letter Z.

- 2nd through 8th character: a – z, A – Z, 0 – 9, _ (underscore), and - (hyphen).

  Note: The DTL compiler does not support the _ (underscore) and - (hyphen).

- No DBCS characters.

Dialog variable names are not case-sensitive. For example, Abc and aBC are considered to be the same variable to the Dialog Manager.

Dialog variable values can range in length from 1 to 32K bytes.

For dialog variables that are defined with the *number-of-occurrences* parameter on the VDEFINE service call, the dialog variable name may be specified with a subscript, using the name(subscript) format on the VCOPY and VREPLACE service. The subscript is an index into a particular element of the array dialog variable. The following rules are observed:

- Regular variable naming conventions apply for the name portion.
- The subscript must be in the range of 1 to 65535 and may be specified as a number or as a scalar dialog variable whose value is a signed 4-byte binary integer dialog variable. For example, either ABC(5) or ABC(index) may be used.
- If the dialog variable is specified without subscript, the first element of the array dialog variable is implied.

Dialog Variable Data Formats

The Dialog Manager supports several data formats for explicitly defined variables:

- CHAR — Character string up to 32K in length
- FIXED — A binary integer
- FIXEDS — A signed integer
- FIXEDU — An unsigned integer
- BIT — A bit string of zeros and ones
- HEX — A hexadecimal string of 0 – 9 and A – F
- DBCS — Double-byte characters
- MIXED — Mixed double-byte and single-byte characters
- BINSTR — A null terminated string
- PACK — Packed decimal
- FLOAT(n) — Floating point number

For more information on these data formats, see "VDEFINE" on page 14-34.

The Dialog Manager converts a dialog variable between the defined variable format and the character format in two situations:

- When a VCOPY or VREPLACE service is used on an explicitly defined variable
- When a variable is retrieved from the dialog variable pool and displayed on a panel or in a message.

When converting a numeric variable to character format, the Dialog Manager produces a character string as follows:

Numeric value	Result
Positive integer	[. . .d]d or +[. . .d]d
Negative integer	−[. . .d]d
Positive rational	[. . .d]dpd[d. . .] or +[. . .d]dpd
Negative rational	−[. . .d]dpd[d. . .]

where *d* is a digit (0−9) and *p* is a period (.).

When converting a character string to an internal numeric format, the Dialog Manager recognizes strings of the following form:

```
[±][. . .d][p[d. . .]]
```

or

```
[±][[...]dt][[d]d]d[p[d...]]
```

where *d* is a digit (0−9), *p* is a period, and *t* is a comma.

For data formats that the Dialog Manager does not directly support, DM applications should rely on native language conversions wherever they are available to convert unsupported data formats into those that the Dialog Manager supports.

Note: The Dialog Tag Language provides facilities to define the appearance and validation checking of variables that are used in panel definitions. For example, numeric values may be translated between the internal storage format and the national language-specific external format by using the TYPE = 'NUMERIC' attribute of the VARCLASS tag. Refer to the VARCLASS and VARDCL tags in the *Dialog Tag Language Guide and Reference* for more information.

System Variables

System variables are used to communicate special information between a DM application and the Dialog Manager. Some of these variables must be defined by the DM application, while others are owned and maintained by the Dialog Manager on behalf of the DM application. System variables defined by an application reside in the dialog variable pool. System variables owned and maintained by the Dialog Manager are kept in an area separate from the dialog variable pool. Copies of these variables can, however, be obtained by the DM application through the use of the VCOPY service.

There are several different types of system variables. A system variable's type identifies its usage and owner. The different types of system variables are input, output, user-modifiable, and non-modifiable. The following list describes the characteristics of each type of system variable:

Input
Input system variables are owned by the DM application and must be defined explicitly by the VDEFINE service or implicitly by the VREPLACE service. Because these variables reside in the dialog variable pool, they must be defined in every dialog variable pool the DM application creates so that the Dialog Manager can access them.

Output
Output system variables are owned by the DM application. The Dialog Manager does not read the values of these variables, but only sets them to provide information to the application. They may be explicitly defined by the application, or they will be implicitly defined by the Dialog Manager after a panel is displayed. If these variables are implicitly defined by Dialog Manager, their values may be accessed using the VCOPY service.

An example of an output system variable is ZCURPOS. This variable is not created until the first panel in a DM application is displayed.

User-modifiable
User-modifiable system variables are owned and maintained by the Dialog Manager but may be modified by the user. These variables are always available and can be modified by issuing certain Dialog Manager system commands, including the CHGDEFS command.

Non-modifiable
Non-modifiable system variables are owned and maintained by the Dialog Manager. These variables are always available for use by the DM application. They cannot be modified by the user or the DM application.

See Chapter 16, "System Variables" on page 16-1 for more information about system variables.

As previously mentioned, some system variables must be defined by the DM application while others are owned and maintained by the Dialog Manager. The Dialog Manager does not prevent an application from defining dialog variables with the same name as the system variables owned by the Dialog Manager. Applications should not do this because the application's copy of the variable is not the one used to control processing or to communicate information between the DM application and the Dialog Manager. This problem is demonstrated in the following example:

An application defines ZPANELID and assigns it a value of 1. The Dialog Manager copy of this system variable has a value of 0. The application displays a panel containing a DTAFLD whose DATAVAR is ZPANELID. When the panel is displayed, ZPANELID is set to 1. However, the panel identifier is not displayed because the system variable used to control the setting of the panel identifier is owned by the Dialog Manager and has a value of 0. This is because panel values are retrieved from the dialog variable pool.

Chapter 8. Using Library Services

DM applications use library support provided by Dialog Manager to define where the dialog elements for an application exist.

Dialog Elements and Library Definition

Some of the dialog elements are categorized as two library types: LIBRARY and HELP. For these library types, you use the LIBDEF command to define libraries by listing the library file names.

The dialog elements that can be defined with the LIBRARY type are application panels, messages, key mapping lists, and icons. To access these dialog elements, the Dialog Manager searches for files in the current directory and the paths specified in the DPATH environment variable.

The dialog elements that can be defined with the HELP type are help panels. To access these dialog elements, the files are searched for in the current directory and the paths specified in the HELP environment variable.

In addition to the LIBRARY and HELP type dialog elements, library services access other dialog elements, such as application command tables. An application command table should have the file name xxxxCMDS.DTL, where xxxx is an *application-id*. The application command table does not need to be defined by the LIBDEF command, since the file name of the dialog element is defined to the Dialog Manager using the application ID and the predefined file suffix. The Dialog Manager searches for the application command table in the current directory and in the paths specified in the DPATH environment variable.

Using the LIBDEF Command

There are two ways to issue LIBDEF commands:

1. LIBDEF commands in the application library definition file
2. LIBDEF service call.

Application Library Definition File – xxxxLIB.APP

The application library definition file can contain LIBDEF commands that define the LIBRARY type and HELP type libraries.

At every DMOPEN, the Dialog Manager searches the current directory and the paths specified in the DPATH environment variable for the application definition file, unless the application ID and instance ID are shared with the previously issued active DMOPEN service call. If the file is found, the Dialog Manager reads the file and uses the library definition as the application default. If the file is not found, the libraries are not defined until a LIBDEF service call is issued from the application program.

To create an application library definition file, you use a text editor. The name of the file must be xxxxLIB.APP, where xxxx is an *application-id*. The syntax of the LIBDEF command is exactly the same as that of the LIBDEF service call.

The application library definition file is an optional file. However, if the file exists and any of the LIBDEF commands in the file has incorrect syntax, the Dialog

Manager issues return codes indicating that an error has occurred during the DMOPEN service call.

Using the LIBDEF Service

The LIBDEF service allows the application to define libraries dynamically. With this service call, you can replace the LIBDEF of the application library definition file.

Each new LIBDEF request replaces the previously issued LIBDEF request of a specific library type. Using the COND key word on the service call, you can define libraries only if no library was already defined by a previous LIBDEF service call or LIBDEF command in the application library definition file.

Syntax and Example of LIBDEF Command

The syntax of the LIBDEF command is:

```
LIBDEF LIBRARY LIBLIST (filename, filename, filename)
LIBDEF HELP LIBLIST (filename, filename, filename)
```

Here are examples of LIBDEF commands:

```
LIBDEF LIBRARY LIBLIST (MYLIB.DTL, SAMPLE.DTL, TESTLIB.DTL)
LIBDEF HELP LIBLIST (MYHELP.HLP, SAMPLEH.HLP, TESTHLP.HLP)
```

When the LIBDEF command is issued, the list of library names enclosed in parentheses is not validated. A non-existent library or invalid file name will be detected later when a dialog element in the library is requested.

For a complete description of the LIBDEF command syntax, see "LIBDEF" on page 14-21.

Chapter 9. Using Dialog Manager Help

Because users will occasionally require additional information about choices, entry fields, messages, or how to proceed with an application that you cannot include on an application panel you should provide associated help panels that the user can access. This chapter describes how to provide help that explains your DM application.

You can provide help for:

- An entire application panel
- A specific field, or item, on an application panel
- Commands
- Messages
- Function keys
- One or more words on a help panel (reference phrases or *hypertext*)
- Help index topics.

The user can also access an application-supplied tutorial if the **Tutorial** choice is listed on the Help pull-down and a tutorial is available. This choice appears on the Help pull-down of a help panel if the TUTORIAL attribute was specified on the HELP tag.

Accessing Help for an Application

Users can access help for an application in the following ways:

- Requesting help
  - By selecting the **Help** pushbutton with the mouse or pressing the function key to which the Dialog Manager HELP command is mapped in the DM application. The Common User Access guidelines recommend that the DM application map the HELP command to F1, which is the Dialog Manager default.
  - By typing HELP in the command entry field.

    Selecting help displays help text for an item on a panel (contextual help), or for an entire panel (extended help). Contextual help gives help information that is specific to the item on which the cursor is positioned when help is requested. If a help panel has not been defined for the item, help information for the entire panel (extended help) is displayed.

    Typing and entering HELP in the command entry field or selecting the **Help** pushbutton while the cursor is in the empty command entry field, displays the list of commands for the application (command help). See "Command Help" on page 9-3 for more information.

    If the user types HELP, then moves the cursor from the command entry field to another field, or item, before selecting Enter, the help panel that is displayed is:

    — Contextual help, or;
    — Extended help, if contextual help was not defined for the item.

    If neither contextual nor extended help are defined, a message displays indicating that help is not available.

- Selecting one of the choices on the action bar Help pull-down of the application panel, if a Help pull-down exists.
- While viewing a help panel, selecting additional help from the Help pull-down or pressing the function key that is assigned to any of these types of help. This help would include extended help, keys help, help for help, help index, and, if defined, a tutorial.
- Selecting a reference phrase, if one is defined, while viewing a help panel.

Types of Help Information

The types of help provided by the Dialog Manager are listed below. They are discussed in more detail on pages 9-3 through 9-5.

Extended help
General help information for the entire application panel

Contextual help
Help for a specific item on the application panel, based on the location of the cursor when the user requests help

Command help
Help for the application-defined commands and the DM commands

Keys help
Help for the application function keys

Reference phrase help
Additional help for one or more selectable words on a help panel

Help index
Help topics listed in an index

Message help
Help information for a displayed message in a message pop-up window

Help for help
A description of the types of help and how to access them

Tutorial
A link provided by the Dialog Manager to a tutorial supplied and run by the application.

You specify the types of help your application will provide when you define the application panels. For example, if you specify the name of a help panel on the HELP attribute of the PANEL tag, you are identifying an extended help panel. If you specify the name of a help panel on the HELP attribute of a tag for an item, or field, on the application panel, such as an action bar choice or a data field, you are identifying a contextual help panel.

The *Dialog Tag Language Guide and Reference* contains a complete description of the tags and how to define the various kinds of help. Once you compile your help panels, you store them in HELP type libraries. The Dialog Manager searches for this library type to access the help panels used in your DM application. For more information on specifying HELP type libraries to the Dialog Manager, see Chapter 8, "Using Library Services" on page 8-1.

Extended Help

Extended help provides general information for an entire application panel. This help information could be an overall explanation of items on the panel, the panel's purpose in the DM application, and what the user can do to interact with the panel. When you define the application panel with the DTL tags, you must specify the name of the help panel using the HELP attribute on the PANEL or PANDEF tag.

To access extended help, the user can:

- Select the **Extended help** choice, if present, from the Help pull-down on the application panel
- Type EXHELP in the command entry field.
- Select the pushbutton on the application panel with the mouse or press the function key to which the Dialog Manager EXHELP command is mapped in the DM application.
- Select the **Extended help** choice from the Help pull-down on a help panel.
- While viewing a help panel, press the function key, F2, to which extended help is mapped.
- Request help on an item that does not have an associated contextual help panel.

Contextual Help

Contextual help provides information specific to an item on an application panel on which the cursor is positioned. This help is contextual because it provides information about the item as it is currently used. The help information should describe the purpose of the item and how the user interacts with it. The item could be:

- An action bar choice
- A pull-down choice
- An input data field
- A selection field
- A selection field choice
- A selection list
- A message
- A user control
- A list field
- An input list column.

Specify the name of the help panel using the HELP attribute on the tag you use to create the field, or item, such as the ABC tag for action bar choices or the DTAFLD tag for data fields. If the item on which help was requested does not have a help panel defined, the Dialog Manager displays the extended help panel. If the user requests help for either the system menu or a default system menu pull-down choice, the Dialog Manager displays a system-provided help panel describing the purpose of the item selected.

Command Help

Command help provides a list of the commands that the user can enter in the application's command entry field. The user can scroll the list and display help for any command in the list. You define the application commands for which help is available using the ICMD tag. You can define a help panel to describe each application command and define the association between the command and the help panel by specifying the ICMD tag within the HELP tag. Refer to the *Dialog Tag*

Language Guide and Reference for more information about defining help for commands.

To access help for commands, the user places the cursor in the command entry field and requests help.

- If the user types a valid command in the field and then requests help, the help panel defined for that command is displayed.
- If the command entry field does not contain a command, or if it contains a command that the Dialog Manager does not recognize (the command was not defined with the ICMD tag), a panel containing an alphabetically sorted list of all the commands is displayed. Included in this list are the DM commands. The user can then select a command from the list and display its help panel. If the user selects a DM command from the list, the Dialog Manager displays its own predefined help panel for the command. The Dialog Manager inserts the DM commands when the commands are displayed. The mouse user can move the pointer to a command and double-click the select button.

Keys Help

You must define the keys help panel or panels. You can use any tags available for a help panel when you define the keys help panel. You could provide a help panel that lists each key and a brief description of each. You can also use reference phrase help on the keys help panel to provide a help panel for each key.

A request for keys help displays the help panel that you name in the ZKEYHELP system variable. You must replace this system variable's value every time that you want the keys help panel to change. See Chapter 16, "System Variables" on page 16-1 for more information.

If a help panel is not provided, the message, "Help for function keys is not available," is displayed when the user requests keys help.

To access the keys help panel, the user can:

- Select the **Keys help** choice, if present, from the Help pull-down on the application panel.
- Enter the KEYS command in the command entry field.
- Press the key to which the KEYS command is mapped.
- Type HELP in the command entry field, then move the cursor to a function key area pushbutton and press the Enter key.
- Select the **Keys help** choice from the Help pull-down on a help panel.

Reference Phrase Help

When defining the source file for a help panel, you can use the RP (reference phrase) tag to identify one or more words for which you want to provide additional information. These words are called reference phrases and are displayed in the color defined for highlighting help text. When the user moves the cursor to a word within the reference phrase, the text and background colors for that word are switched.

Specify the name of the help panel using the HELP attribute within the RP tag. Reference phrases can be used only on help panels. They can be coded anywhere within the information region of a help panel.

The first reference phrase on a help panel will have the input focus when the help panel is displayed. Pressing the Enter key displays the help panel for that reference phrase. The mouse user can move the pointer to the reference phrase and click the select button. If there is more than one reference phrase on a help panel, the user can press the Tab key or use the mouse to move the input focus to the next reference phrase.

Help Index

Help index provides a list of all help topics in the application. The user can search for specific help topics or view the entire index. To include the help topic index in your application, you must use the ITOP tag in your help panel definitions. The ITOP tag defines the help topic name that will appear when the indexed topics are presented for selection. You can also use the ISYN tag, which defines synonyms that will be searched to locate specific help topics.

To access the index, the user can:

- Select the **Help index** choice from the application panel's Help pull-down
- Enter the INDEX command in the command entry field
- Select the pushbutton on the application panel with the mouse or press the function key to which the INDEX command is mapped
- Select the **Help index** choice from the Help pull-down on a help panel
- While viewing a help panel, press the function key (F11) to which the help index is mapped.

Message Help

Message help provides help for messages displayed in Dialog Manager message pop-up windows. Use the MSG tag when defining a message. You must specify the *help-panel-name* on the HELP attribute of the MSG tag when providing a help panel that gives the user additional information about the message. To access message help, the user can request help by pressing the Help key or selecting the Help pushbutton with the mouse, while the message is displayed.

Help for Help

Help for help provides general information about using and accessing the various types of help. It displays when the user selects the **Help for help** choice from the Help pull-down on a help panel or presses the Shift and F10 keys.

Tutorial

You can define a tutorial for your application with DTL tags. When the user requests the **Tutorial** choice from the help pull-down, or using the Shift and F2 keys when the help window is active, the Dialog Manager runs the TUTORIAL command from the application command table. The application can then handle the command by running the appropriate tutorial program, which the application has provided.

If you specify a tutorial name on the TUTORIAL attribute of a HELP tag and the user selects the **Tutorial** choice from a help panel, the TUTORIAL command executes from the application command table. When you define the TUTORIAL command in the application command table as a SETVERB or PASSTHRU, the name specified on the TUTORIAL attribute will be passed back as the parameter.

- If you define the TUTORIAL command as a SETVERB, ZVERB will contain the keyword TUTORIAL, and the name of the tutorial will be placed in ZCMD.

- If you define the TUTORIAL command as a PASSTHRU, ZCMD will contain the keyword TUTORIAL followed by a blank and the name of the tutorial.

If you do not define the TUTORIAL command in the application command table, the default Dialog Manager processing occurs. Refer to the CMD, CMDACT, and CMDTBL tag descriptions in the *Dialog Tag Language Guide and Reference* for information about defining the TUTORIAL command. See "Defining the Command Action" on page 6-2 for information about the SETVERB and PASSTHRU commands.

Help Window and Panel

When a user requests help, a main help window is displayed. Each help panel is displayed inside a help text window. Once a help window is displayed on the screen, the user is in control of the window's size and location. When the user sizes the help text window, text reformats to fit the new size while the help window is displayed. All subsequent help panels are placed in the help text window.

The Action Bar

The main help panel has an action bar located below the title bar. Selecting a choice on the action bar causes the associated pull-down to display. The following sections describe each pull-down.

Services

The **Services** menu lets the user locate, print, or copy information without leaving help. The **Services** menu contains the following choices:

Print Allows the user to print the current helps being viewed, print all the helps for the current application, print selected helps, print the help index, or print the help contents.

The text that the user wants to print is submitted to the print spooler for printing so that the user can continue with other tasks.

- The **This section** choice is used to print the help panel that the user is currently viewing.
- The **All sections** choice is used to print all of the help panels for the application.
- The **Marked sections** choice is used to print selected help topics. If the user has not selected any topics from the Contents window, this choice is de-emphasized.

  Selections are marked with the mouse by pressing the Ctrl key and pressing mouse button 1. Selections are marked with the keyboard by using the cursor keys to highlight the item and then pressing the Spacebar. The same key sequences are used to toggle the selection.
- The **Index** choice is used to print the help index.
- The **Contents** choice is used to print the help table of contents.

Copy Copies the help that the user is currently viewing to the system clipboard.

Copy to file Copies the current help to the file TEXT.TMP in the user's working directory. The user can then use an editor to modify the file or embed the file in another file.

Append to file Appends the current help to the end of the TEXT.TMP file in the user's working directory. This allows the user to accumulate selected helps.

Options

The **Options** menu provides many choices. The Dialog Manager supports the following choices:

Contents Displays the help index Contents window. The Contents window is generated from the help panel title. The Contents window is opened inside the main help window and contains a table of contents with a list of help panel titles. You can select a help panel from this list.

A scroll bar is always displayed in the Contents window. If the Contents listing is not scrollable, an alarm sounds when the user clicks on the scroll bar.

Viewed pages Lists the help panels the user has viewed. This choice provides a list of all the help panels viewed during the current help session.

Previous help (Esc) Removes the current help panel and displays the help panel underneath it, if one exists.

Help

The functions associated with the choices on the Help pull-down are performed automatically. The function key assigned to each choice also appears on the help panel's Help pull-down. While viewing a help panel, the user can display additional help by either selecting a choice from the Help pull-down or by pressing the function key that is assigned to the type of help desired. While viewing a help panel, the user can display another help panel by either selecting a choice from the Help pull-down or by pressing the function key that is assigned to the type of help desired. A description of the Help pull-down follows:

Help for help (Shift + F10) Gives general information about the types of help that are provided and how to display the help.

Extended help (F2) Displays the extended help panel for the current application panel. See "Extended Help" on page 9-3.

Keys help (F9) Displays the keys help panel defined with the ZKEYHELP system variable. See "Keys Help" on page 9-4 for more information about keys help.

Help index (F11) Displays the help index. See "Help Index" on page 9-5.

Tutorial (Shift + F2) Optional. Requests that the application-supplied tutorial be run.

Help Function Keys

Certain function keys are always active (the number and nature of these keys are environment-specific) independent of the current application. Other function keys are application-defined and are active or inactive as defined by the selected application. When a help panel is displayed on the screen, only those keys that the OS/2 Information Presentation Facility defines, plus the keys that function independently of the application (such as the task-switch key), are active.

The active function keys on a help panel include:

- Esc to remove the current help panel and display the previous one
- F2 for the **Extended help** choice
- Alt and F4 to remove the help window
- F9 for the **Keys help** choice
- F10 to move the cursor to and from the action bar
- F11 for the **Help index** choice
- Shift and F10 for the **Help for help** choice
- Shift and F2 for the **Tutorial** choice.

The Esc key removes the current help panel and displays the previous one. If only one help panel has been displayed in the help window, the Esc key removes the help window. Application-defined function keys are ignored while the help window is the active window.

Interacting with the Help Window

Displaying the Main Help Window

The first time the user requests help on information in a primary or pop-up window and a help window is not on the screen for that application, the OS/2 Information Presentation Facility creates the main help window. The help panel is displayed in a help text window within the main help window.

Replacing an Existing Help Panel

If a help text window already exists for the application, the next help panel overlays the existing help panel in that window. The help panels are stacked inside the help window, one help panel on top of the previous one, until the help window is removed.

Removing Help Windows

An explicit user action, such as pressing the Alt and F4 keys, is required to remove the help window. Pressing Esc removes the current help panel and displays the previous one. If only one help panel has been displayed, pressing Esc removes the main help window. The Dialog Manager also removes the help window when the associated window is removed.

Sizing Help Windows

After the help window is displayed, users can control the position and size of both the main help window and the help text window. They can make the width and depth of the window larger or smaller, or reduce it to its minimum size. When the help text window is sized, the text reformats to fit the new size.

Help panels include the maximize icon, so that the user can maximize the help window to the size of the screen. The main help window cannot be reduced to an icon.

Chapter 10. Compiling, Linking, and Running Dialog Manager Applications

This chapter lists the language-specific requirements for compiling, linking, and running DM applications that are written in the following languages:

- IBM C/2™ Version 1.1
- IBM COBOL/2™
- IBM FORTRAN/2™ Version 1.02
- IBM Macro Assembler/2™
- IBM Pascal/2™
- Procedures Language 2/REXX.

Hereafter, these languages will be referred to, respectively, as C, COBOL, FORTRAN, MASM, Pascal, and Procedures Language.

This chapter also suggests methods you can use to maximize the performance of your DM application when using the Dialog Manager.

For an overview of the steps required to build Dialog Manager programs and applications, refer to the *Building Programs* manual.

Compiling Requirements

For each of the supported languages, the Dialog Manager provides a special include file that you can include in the application program's code. If you include this file, your application will automatically contain the DM procedure declarations and a declaration of the DM communication area. If you do not include this file, you will need to add this information to your application code.

C

Dialog Manager provides the special include file called ISPCAST.H for C applications. This include file takes care of the typecasting necessary for use by small and medium program models. The include file also provides a type declaration for the DMCOMMBLOCK parameter, which specifies the DM communication area structure.

Assuming they are written in C, user control source files must be compiled with a customized storage model described in the C/2 *Compile, Link, and Run* manual. One of the following must be specified:

- The /Azzu option
- The /Azzw option, and the "_loadds" keyword added to the control procedure function.

Note: "zz" specifies the compilation model. Refer to "Creating Customized Storage Models" in the C/2 *Compile, Link, and Run* manual.

C/2, COBOL/2, FORTRAN/2, Macro Assembler/2, and Pascal/2 are trademarks of the International Business Machines Corporation.

In addition, when you compile user control source files, you must specify the -GS option to disable stack checking. See Chapter 12, "Additional Features of the Dialog Manager" on page 12-1 for more information on user controls.

Note that many of the C/2 library functions are not reentrant and special precautions may be necessary in order to use them within a user control. Refer to the C/2 *Compile, Link, and Run* manual for more details.

COBOL

Dialog Manager provides the special include file called ISPCOBOL.INC for COBOL applications. The /LITLINK option must be specified.

Dialog Manager is not supported for COBOL small or compact program models; therefore, do not use the /MODEL"COMPACT" or /MODEL"SMALL" options when compiling.

FORTRAN

Dialog Manager provides the special include file called ISPFORT.INC for FORTRAN applications. Dialog Manager does not require any special commands when compiling a FORTRAN application.

Macro Assembler

Dialog Manager provides the special include file ISPCALL.INC for MASM applications. To use this file in your application, specify

```
INCLUDE ISPCALL.INC
```

This file provides a struc definition of the DM communication area and ensures that the far calls are made to the Dialog Manager routines.

Pascal

Dialog Manager provides the special include file called ISPPASC.INP for Pascal applications. This file provides a TYPE definition for the DM communication area and supplies definitions for the Dialog Manager ISPCI and ISPCI2 service calls.

Linking Requirements

When linking a DM application, you must use the /STACK option to allow enough stack space for the Dialog Manager to operate. A minimum starting value of /STACK:20000 is recommended. This value applies to all languages, and may increase or decrease based on the application's size and complexity.

You can specify the /STACK option on any LINK command or place the /STACK option in a module definitions (.DEF) file. For information about using the LINK command, consult the compile, link, and run documentation for the language you are using.

A DM application is also a Presentation Manager application. Therefore, a DM application needs to run in a Presentation Manager window. To enable your DM application to run in a windowing environment, you need to link with a module definitions (.DEF) file, which contains the statement:

```
NAME modulename WINDOWAPI
```

where *modulename* is the name of your module or application. The WINDOWAPI attribute is required.

C, FORTRAN, Macro Assembler, Pascal

The libraries to link to with each language are:

Language	Library
C	C_ISP.LIB
FORTRAN	FO_ISP.LIB
MASM, Pascal	P_ISP.LIB

COBOL

For COBOL applications, the library to link to depends on the program model being used.

The specific library or object module to use with each program model is given by the following chart:

COBOL Model	Library
Huge	CO_ISP.LIB
Large	CO_ISP.LIB
Medium	CO_ISPM.LIB

In addition to the preceding restrictions, when linking programs compiled with the COBOL/2 compiler for medium program models, you must specify the /DO option.

Running Requirements

Procedures Language

In Procedures Language, a DM application is considered to be a Presentation Manager program. Therefore, a DM application needs to run in a Presentation Manager window. To enable your DM application to run in a windowing environment, you need to initiate the application with the following statement:

```
PMREXX programname
```

where *programname* is the name of your application.

Maximizing the Performance of Your DM Application

The following list contains suggestions you can follow to maximize the performance of your DM application.

- Increase the DISKCACHE parameter in CONFIG.SYS to increase the size of your machine's disk caching buffer. This reduces disk I/O by keeping portions of frequently referenced files in memory.

Note: Increasing DISKCACHE reserves more memory for the system and reduces the memory available to OS/2 applications. Consider the amount of available memory before increasing the DISKCACHE parameter.

- Use the minimum number of DM libraries. DM must open and search for the dialog elements it needs in each of the libraries specified in the LIBDEF service. By reducing the number of libraries, you reduce the time DM spends searching for the required dialog elements. Specifying the same target library when compiling your source GML files will result in the minimum DM run-time search. Refer to the chapter on compiling GML source files in the *Dialog Tag Language Guide and Reference* for more information.
- Put your DM libraries on a virtual disk. You can define a virtual disk using the VDISK device driver in your CONFIG.SYS. A virtual disk stores data in the computer's memory which can be accessed much faster than from a fixed disk.
- Put the DM libraries in the current directory or as the first items in the list of directories specified in the DPATH environment variable. DM searches the current directory and then the directories listed in the DPATH environment variable for the DM libraries and application library definition file. You can reduce DM search time for these files by listing them first in the search sequence.
- Avoid unnecessarily complex panels. The time required to display a panel is directly related to its complexity. When designing your application, keep in mind that a panel with many fields requires more time to display. If you reduce the complexity of a panel, it will display more quickly.

Chapter 11. Installing Dialog Manager With Your DM Application

When you develop and compile a DM application, you have available to you the Dialog Manager run-time files that are installed on your system as part of the OS/2 Programming Tools and Information Version 1.2 installation process. However, you may want to make your DM application available to users who have not installed Dialog Manager, or who have a different version of the Dialog Manager installed on their systems. To do this, you need to create one or more application diskettes that contain the appropriate Dialog Manager run-time files and DM application files so that users can install all the files necessary to run your DM application.

This chapter provides information on creating application diskettes that contain the application-specific and Dialog Manager run-time files required to run your DM application. In addition, this chapter describes the installation utility, DLL and Data File Installation Utility, provided in the OS/2 Programming Tools and Information Version 1.2 which allows users to manage the installation of the Dialog Manager run-time files on their systems. Finally, this chapter suggests several ways to set the OS/2 DPATH and HELP environment variables to reference the appropriate Dialog Manager run-time files.

Creating an Application Diskette

An application diskette contains all the files necessary to install and run your application on a user's system. In addition, an application diskette contains an installation program that you write to ensure that the files are installed correctly on a user's system. Although some application diskettes may contain additional files, all application diskettes should contain the following:

- Dialog Manager application-specific files
- Dialog Manager run-time files
- DLL and Data File Installation Utility
- Application installation program.

Each of these application diskette components is discussed below.

Dialog Manager Application-Specific Files

When you create the dialog elements (panels, key mapping lists, messages, and command table) and write the program code (using Dialog Manager services) to support your application, you have the flexibility of storing these files in the directory of your choice.

Dialog Manager Run-time Files

There are two types of Dialog Manager run-time files that you need to include with your DM application on an application diskette: dynamic link library files (with an extension of .DLL) and machine-readable information (MRI) files (with extensions of .HLP and .DTL). The Dialog Manager run-time files have extended attributes, which the DLL and Data File Installation Utility uses to identify and track the versions of the run-time files during the installation process. Because of environment differences, you must *not* copy, move, or otherwise manipulate the Dialog Manager run-time files in the PC/DOS environment. Doing so may alter the

extended attribute information of these files and make them unusable by the DLL and Data File Installation Utility.

Dialog Manager .DLL Files

When the Dialog Manager is installed on your system as part of the OS/2 Programming Tools and Information Version 1.2 installation process, the .DLL files are installed in the fixed directory, TOOLKT12\DLL. The files included in this directory are listed below:

```
TOOLKT12\DLL
------------
   EGIDMC1.DLL          EGIYERR.DLL
   EGIDMN1.DLL          EGIYCBMP.DLL
   EGIDMN2.DLL          EGIYCBSP.DLL
   EGIDMP1.DLL          EGIYCISP.DLL
   EGIDMS1.DLL          EGIYFISP.DLL
   EGIDMV.DLL           EGIYPISP.DLL
   EGIDMX1.DLL          ISPCIR.DLL
   EGIZNLS.DLL          ISPMSGDB.DLL
```

You must copy these files to your application diskettes from the TOOLKT12\DLL directory installed on your system.

Dialog Manager MRI Files

In addition to the .DLL files installed on your system, Dialog Manager MRI files are installed on your system during the OS/2 Programming Tools and Information Version 1.2 installation process in the TOOLKT12\DM directory. The files in this directory are listed below:

```
TOOLKT12\DM\ISPHELP.HLP
TOOLKT12\DM\ISPLIB.DTL
```

You must copy these files from the TOOLKT12\DM directory installed on your system to your application diskettes.

Note: If you are using a multiple-language version of the OS/2 Programming Tools and Information Version 1.2 to develop your applications, you have twelve Dialog Manager MRI directories on your system. Each of these directories contains the same files as those previously listed, translated for the language indicated by the two-character language ID, *xx*, in the directory name. For example, the directory TOOLKT12\DM\FR contains the French MRI files and the directory TOOLKT12\DM\GR contains the German MRI files. You must ensure that the language-specific MRI files you use to develop your DM application are copied to your application diskette.

DLL and Data File Installation Utility

The OS/2 Programming Tools and Information Version 1.2 provides the DLL and Data File Installation Utility that you must use to install the Dialog Manager run-time files on the user's system. Because DM applications could be developed using different versions of the Dialog Manager run-time files, the DLL and Data File Installation Utility ensures that the latest version of these files is installed.

The DLL and Data File Installation Utility includes the following files:

```
TOOLKT12\BIN\INSTTOOL.EXE
TOOLKT12\BIN\INSTTOOL.MSG
```

Copy these files to your application diskettes so that they can be used during the installation process.

Application Installation Program

An application installation program should contain the appropriate statements to install all the files required to run your DM application. Each application has different installation requirements.

Writing an Application Installation Program

Once you have copied the Dialog Manager run-time files, your application-specific files, and the DLL and Data File Installation Utility to your application diskettes, you must provide a program that users can access to install the Dialog Manager run-time and application files. You determine the content of the program used to install the necessary files on the user's system. The following sections provide some guidelines to follow when writing your application installation program.

Installing Application-Specific Files

As the developer of a DM application, you are responsible for ensuring that all files required to run your application are included on the application diskettes and that users can access these files so that your application runs without error. Your installation program must include all the files that make up your DM application.

Installing Dialog Manager Run-Time Files

Your installation program should install the Dialog Manager run-time files on the user's system so that the correct files can be accessed when running your application. The following sections provide suggestions for installing the Dialog Manager run-time files so that you can take advantage of the DLL and Data File Installation Utility and also control access to the MRI files. You must follow these guidelines to ensure that your application can coexist with other DM applications.

Specifying Directories for Dialog Manager .DLL Files

Although your DM application can install the Dialog Manager .DLL files in any directory you choose, you should install the files in a fixed subdirectory off the directory where PMWIN.DLL resides. This directory is called C:\OS2\DLL\SHARED. The advantages of installing the files in this directory include the following:

- The amount of time the DLL and Data File Installation Utility requires to search the LIBPATH is reduced. The DLL and Data File Installation Utility uses the LIBPATH, DPATH, and HELP values as defined in the CONFIG.SYS file.
- Only the first application installed needs to add the directory to the LIBPATH and reboot the system. If applications using later versions of the Dialog Manager .DLL files are installed, these files are merely replaced in the fixed directory.
- The DLL and Data File Installation Utility creates this fixed directory to ensure that only one version of the run-time files exists on the system and that these files are the most current version.

Specifying Directories for Dialog Manager MRI Files

As discussed previously, when the OS/2 Programming Tools and Information Version 1.2 is installed, the Dialog Manager MRI files are stored on your system. If you have an English version of the OS/2 Programming Tools and Information Version 1.2 installed on your system, the Dialog Manager MRI files are stored in the directory TOOLKT12\DM. If you have a multiple-language version of the OS/2 Programming Tools and Information Version 1.2 installed on your system, you have twelve language-unique Dialog Manager MRI directories. These directories are:

TOOLKT12\DM\DA Danish

TOOLKT12\DM\EN English (UK uses this MRI)

TOOLKT12\DM\FC French Canadian

TOOLKT12\DM\FI Finnish

TOOLKT12\DM\FR French

TOOLKT12\DM\GR German

TOOLKT12\DM\IT Italian

TOOLKT12\DM\NL Dutch (Netherlands)

TOOLKT12\DM\NO Norwegian

TOOLKT12\DM\PT Portuguese

TOOLKT12\DM\SP Spanish

TOOLKT12\DM\SV Swedish

Your application must install these files into the C:\OS2\DM*xx* directory, where *xx* is the two-character ID for the language. By using this directory-naming convention when installing Dialog Manager MRI files, you can accommodate the following situations:

- Allowing the coexistence of multiple language MRIs on one system.

  The file names of DM MRIs are fixed, regardless of national languages. When you use the DLL and Data File Installation Utility, these files will be installed in the named directory even if files with the same names exist in a different directory.

- Ensuring the existence of a single copy of a given language MRI on one system.

  The DLL and Data File Installation Utility installs the MRI files on the named directory. If the directory currently exists, the files are either overlaid or unchanged (depending on the version of Dialog Manager installed).

Example: Installing a DM Application

The following example illustrates the steps used to ensure the proper installation of your English DM application files and Dialog Manager run-time files on a user's system. In the example, the application diskette has the following layout:

```
root directory              /* insttool.exe, insttool.msg     */
\install\app directory      /* DM application-specific files  */
\install\dlls directory     /* DM .DLL files                  */
\install\mris\en directory  /* DM MRI files                   */
```

Step 1: Choose an appropriate method to copy the application-specific files to the user's system.

Step 2: Create lists for the fully-qualified DM run-time file names and MRI file names. This list must begin with a comment line whose first character is a semicolon (;).

dmdll file

```
;comment
a:\install\dlls\EGIDMC1.DLL
a:\install\dlls\EGIDMN1.DLL
a:\install\dlls\EGIDMN2.DLL
a:\install\dlls\EGIDMP1.DLL
a:\install\dlls\EGIDMS1.DLL
a:\install\dlls\EGIDMV.DLL
a:\install\dlls\EGIDMX1.DLL
a:\install\dlls\EGIZNLS.DLL
a:\install\dlls\EGIYERR.DLL
a:\install\dlls\EGIYCBMP.DLL
a:\install\dlls\EGIYCBSP.DLL
a:\install\dlls\EGIYCISP.DLL
a:\install\dlls\EGIYFISP.DLL
a:\install\dlls\EGIYPISP.DLL
a:\install\dlls\ISPCIR.DLL
a:\install\dlls\ISPMSGDB.DLL
```

dmmri file

```
*a:\install\mris\en\ISPHELP.HLP
*a:\install\mris\en\ISPLIB.DTL
```

where an asterisk (*) preceding the pathname indicates that the file must be installed in the target directory. For DM MRI files, the asterisk is required.

Step 3: Use the DLL and Data File Installation Utility to install Dialog Manager run-time files:

```
INSTOOL dmdll c:\OS2\DLL\SHARED    /* install DLL files */
INSTOOL dmmri c:\OS2\DMEN          /* install MRI files */
```

The following is a sample command file that could be used to install your application-specific files and the Dialog Manager run-time files on a user's system. The command file (INSTALL.CMD) is started by typing INSTALL from the [A:\] prompt.

```
@ECHO OFF
Rem  Create a directory for your application
MKDIR C:\dmapp

Rem  Install application-specific files
COPY A:\INSTALL\APP\*.* C:\dmapp

Rem  Use DLL and Data File Installation Utility to
Rem  install the Dialog Manager run-time files
INSTTOOL dmdll C:\OS2\DLL\SHARED
if errorlevel 1 goto ABEND
INSTTOOL dmmri C:\OS2\DMEN
if errorlevel 1 goto ABEND

ECHO Installation completed successfully.
goto EXIT

:ABEND
ECHO Installation failed!

:EXIT
```

Using the OS/2 DPATH and HELP Environment Variables to Control MRI Files

Although only one version of Dialog Manager run-time files can exist on a system, multiple-language versions of DM MRI files can exist on a system. The OS/2 DPATH and HELP environment variables control which versions of MRI files are available to an application. The application can internally modify the OS/2 DPATH and HELP environment variables, or it can provide an installation program to store the environment information with the application's program information. This will allow the user to run multiple DM applications with each application using a different set of MRI files. To achieve this result, code your application to do one of the following:

- The application can internally modify the OS/2 DPATH and HELP environment variables by using DosGetEnv to receive the environment segment and manually adjust these variables. The OS/2 DPATH and HELP environment variables must be established prior to the first DMOPEN call.
- The application can provide an installation program that uses PrfAddProgram to add the application to the OS2.INI file and specify an environment containing the OS/2 DPATH and HELP environment variables which includes the MRI directory.

  Note: The user cannot modify this information through Change Program Information.
- The application can be started and the OS/2 DPATH and HELP environment variables can be established using an OS/2 command file (.CMD). Because CMD runs only in a full-screen or text windowed session, initiating an application using command files requires two sessions to start the application.

Example: Installing Multiple DM Applications

This example describes the steps required to install multiple DM applications (using different language-unique Dialog Manager MRI files) on a user's system. As the example illustrates, only one version of Dialog Manager run-time files is installed, while two versions of Dialog Manager MRI files (English and French) are installed.

Note: The user's system starts without Dialog Manager installed.

Step 1: Installing an English DM application

1. Install the DM .DLL files in the C:\OS2\DLL\SHARED directory.
2. Install the MRI files in the C:\OS2\DMEN directory.
3. When the English application is initialized, set the OS/2 DPATH and HELP environment variables to always include the C:\OS2\DMEN directory before the first DMOPEN call or the English application is started.

Step 2: Installing a French DM application

1. The DLL and Data File Installation Utility ensures that the most current version of the DM .DLL files are placed in the C:\OS2\DLL\SHARED directory.
2. Install the MRI files in the C:\OS2\DMFR directory.
3. When the French application is initialized, set the OS/2 DPATH and HELP environment variables to always include the C:\OS2\DMFR directory before the first DMOPEN call or the French application is started.

DLL and Data File Installation Utility Messages

The DLL and Data File Installation Utility provides messages that address application installation errors. Some of these messages provide information on correcting errors as you write your application installation program. Other messages provide information to the application user concerning how to proceed when an error is encountered in the application installation process.

The following table contains the message numbers and descriptions for the messages that the DLL and Data File Installation Utility returns when it encounters an error. The table also describes actions that the application developer and application user can follow when the error occurs. You may want to include the application-user related messages and actions in online or hardcopy documentation that you provide with your application.

Table 11-1 (Page 1 of 5). Error Messages Returned by DLL and Data File Installation Utility

Number	Description	Action

ITL0002I	`Usage: insttool listfile [targetdir]`
	`listfile - File with the list of files to install, one per line.`
	`targetdir - Target directory to install the files in.`
	
	A list file was not specified or additional parameters (other than *listfile* and *targetdir*) were specified.
		Application Developer: Specify the list file and target directory, if necessary.

ITL0003E	`The target directory could not be created.` A file exists with the same name as the target directory, or the call to create the directory failed. The most common reason that the directory is not created is if the path in which the subdirectory is specified contains a non-existent directory.	Application Developer: Specify a valid target directory. Application User: Rename the file to a name other than the target directory name.
ITL0004E	`The specified list file could not be opened.` The specified list file is not found or is locked by another process.	Application Developer: Specify a valid list file. Application User: Make sure the installation diskette is in the drive. Shut down all other applications.
ITL0005E	`A file specified in the list file could not be found.` An invalid file name is in the list file referenced in the installation program.	Application Developer: Specify a valid file name and ensure that each file specified in the list file exists on the application diskettes. Application User: Make sure the installation diskette is in the drive. Shut down all other applications. Contact your System Administrator or the developer of your application.
ITL0006E	`The LIBPATH could not be found in CONFIG.SYS.` The current libpath cannot be determined because the LIBPATH= statement is not found in CONFIG.SYS.	Application Developer: Restore or update the CONFIG.SYS file so that it contains a LIBPATH= statement. Application User: Restore or update the CONFIG.SYS file so that it contains a LIBPATH= statement.
ITL0007E	`An existing file could not be renamed.` A .DLL or data file is locked by another process.	Application Developer: End the process that locks the file. Application User: Shut down all other applications.

Table 11-1 (Page 2 of 5). Error Messages Returned by DLL and Data File Installation Utility

Number	Description	Action
ITL0008E	A file specified in the list file could not be copied. This occurs if there is a data error on the installation diskette and a new file cannot be copied to the fixed disk or if the fixed disk is full.	Application Developer: Delete some files to allow space for the file to be copied. Ensure that the installation diskette does not contain errors. Application User: Delete some files to allow space for the file to be copied. Contact your System Administrator or the developer of your application.
ITL0009E	An existing file could not be moved. This occurs when there is an attempt to move an existing file to the SHARED directory. The cause may be that the attempted move crosses a volume and the target disk is full.	Application User: Delete some files to allow space for the file to be moved. Contact your System Administrator or the developer of your application.
ITL0010E	A file cannot be renamed to its proper name. This occurs when either a newly installed file or an existing file was moved to the SHARED directory and cannot be renamed to its original name.	Application User: Contact your System Administrator or the developer of your application.
ITL0011E	Cannot update CONFIG.SYS. Either a temporary file that is required to update CONFIG.SYS cannot be opened (the fixed disk may be full), or the existing CONFIG.SYS cannot be renamed to a temporary name (CONFIG.SYS may be locked by another process), or the updated, temporary version of CONFIG.SYS cannot be renamed to CONFIG.SYS.	Application User: Check the available disk space on the drive in which CONFIG.SYS is located, and delete files if necessary to obtain the required space. Make sure that all applications are shut down; then, attempt the application installation process again.
ITL0012I	The specified files were successfully installed. The installation process completed successfully.	No action required. Message is informational only.
ITL0013E	Cannot determine the version of a file. The DLL and Data File Installation Utility cannot query the extended attribute of a file (either the one being installed or one that exists on the fixed disk). This can occur because another process has locked the file, or because the .DLL and data files did not have the correct extended attribute.	Application Developer: Do not use PC/DOS to move or otherwise manipulate files. If the extended attribute has been lost, attempt the installation of the OS/2 Programming Tools and Information Version 1.2 again. Application User: Contact your System Administrator or the developer of your application.

Table 11-1 (Page 3 of 5). Error Messages Returned by DLL and Data File Installation Utility

Number	Description	Action
ITL0014E	`The descriptions of two files with the same name do not match.` This can occur when the DLL and Data File Installation Utility attempts to install a file with the same name as an existing file. If the descriptions of these files do not match, the install fails.	Application Developer: Contact your IBM Service Representative. Application User: Contact your System Administrator or the developer of your application.
ITL0015E	`Incorrect or missing parameters.` This occurs when a user specifies the DLL and Data File Installation Utility syntax incorrectly. An error occurs when the user omits the required *listfile* parameter or specifies invalid parameters (other than *listfile* and *target*). Message ITL0002I, which lists the valid syntax, is also displayed.	Application Developer: Use the correct syntax and attempt the application installation process again.
ITL0016E	`Out of memory.` This occurs when the amount of storage is insufficient to install the application.	Application User: Be sure that there is space available on the drive to swap from virtual to auxiliary storage and that swapping is enabled. Also, shut down other applications to increase the available storage.

ITL0017I	`*** IMPORTANT! ***`
	`The system configuration file CONFIG.SYS was updated.`
	`The system must be rebooted for the changes to take effect.`
	The DLL and Data File Installation Utility issues this message after it automatically updates the CONFIG.SYS file during the application installation.
		Application User: Reboot the system after completing the application installation.

ITL0018I	`The installation did not complete. Any changes have been removed.` This message is displayed whenever there is an error. The DLL and Data File Installation Utility attempts to return the system to its pre-installation state.	Application User: Correct the error that precedes this message and attempt the application installation again.
ITL0019E	`The specified list file does not contain any files to install.` This occurs when the list file to be installed is empty or contains only comment lines.	Application Developer: Specify a list file with valid contents.
ITL0020E	`A target directory must be specified.` This occurs when a file in the list file is preceded by an asterisk, but no directory is specified. When a file in a list file to be installed is preceded by an asterisk, a target directory for installation must be specified.	Application Developer: Specify a target directory in the list file specification.

Table 11-1 (Page 4 of 5). Error Messages Returned by DLL and Data File Installation Utility

Number	Description	Action
ITL0021E	`The shared directory could not be created.` The shared directory is a subdirectory (named SHARED) of the directory in which PMWIN.DLL is found. This directory is usually called C:\OS2\DLL\SHARED. This message can also be generated when the directory cannot be created (for example, if the disk space is full) or when a directory entry with this name is found, but the entry is a file rather than a subdirectory.	Application Developer: Delete some files to allow space for the directory to be created. Application User: Check to see if any of the problems listed occurred and correct the problem. Rename the file with the same name as the target directory.
ITL0022I	`Close down existing applications and try again.` This occurs when a previous operation failed because a .DLL or data file required for installation has been opened by another process.	Application User: After closing down any existing applications, attempt the application installation process again.
ITL0023E	`Installation was interrupted!` This message is generated when the user interrupts the application installation by pressing Ctrl + c or Ctrl + Break; when another process explicitly halts the install program; or when a hard error occurs and the user selects Abort.	Application User: Attempt the application installation process again.
ITL0024I	`Processing file %1.` This message describes the file in which the error occurred, if applicable.	No action required. Message is informational only.
ITL0025E	`Cannot open CONFIG.SYS.` This error occurs when the CONFIG.SYS file has been opened by another process.	Application User: Ensure that the CONFIG.SYS file exists and that no other applications have opened it. End existing applications and attempt the application installation process again.
ITL0026I	`%1 new file(s) installed.` This message indicates the number of new files copied from the installation diskette. A file is copied if there is no existing file with the same name on the fixed disk, or if an existing file with the same name is an older version than the file being installed.	No action required. Message is informational only.
ITL0027I	`%1 existing file(s) moved to a common location.` This message indicates the number of existing files that were moved to the shared directory. An existing file is moved from the hard disk when it is the same or a newer version than a file being installed.	No action required. Message is informational only.

Table 11-1 (Page 5 of 5). Error Messages Returned by DLL and Data File Installation Utility

Number	Description	Action
ITL0028I	%1 old file(s) removed. This message indicates the number of files removed from the fixed disk. An old file is an older version (or a duplicate with the same version) of a file that is being installed or moved to a shared location.	No action required. Message is informational only.

Chapter 12. Additional Features of the Dialog Manager

This chapter describes additional Dialog Manager features you can use when developing your DM application, including forced display exits, user controls, and user exits. For example, if you want your application to allow for extended processing, you should read the section on forced display exits. If you want to add graphics to your DM application, read the section on user controls. If you want to get control of a DM application during panel processing rather than after a DISPLAY service has returned control, read the section on user exits.

The features described in this chapter exploit various functions provided by the OS/2 operating system; thus, additional knowledge of the operating system is necessary to use forced exit display, user controls, and user exits.

Forced Display Exit

In certain circumstances, applications need to run for long periods of time (greater than one second) without requesting information or interactions from the application user. Examples of this are when some long calculation is required or when the application must retrieve data from a database or some remote computer. In the OS/2 windowing environment, the user cannot interact with any other windowed applications while a single-thread application runs. In order to allow the user to interact with other applications in this windowing environment, the Dialog Manager FORCEXIT service allows you to provide a special *forced display exit* from the application.

Beginning the Extended Processing

Applications that find it necessary to perform extended processing should do so in a separate thread. A thread is a separate unit of processing which runs independently from the rest of your application. On the Dialog Manager thread, your application should display a panel that lets the user know that the application is performing extended processing.

When the user requests an action that requires a great deal of processing time (more than one second), your application must do two things:

- Create a separate thread to perform that processing.

  The separate, non-Dialog Manager thread (a thread that does not request Dialog Manager services) is started using the DosCreateThread call. Refer to the *OS/2 System Application Program Interface* for additional information on this call. This thread must not request Dialog Manager services by performing a DMOPEN.

  A non-Dialog Manager thread can use the FORCEXIT service because FORCEXIT is the only Dialog Manager service that does not require a preceding DMOPEN service call in that thread.

  Note: A separate process could also use the FORCEXIT service as long as it can gain access to shared program storage containing the DM communication area so that it can copy the contents of the DM communication area.

- On the Dialog Manager thread, display a panel that lets the user know that your application is processing.

  Use the Dialog Manager DISPLAY service to display a panel.

Note: Your application is responsible for all communication between these threads.

Refer to the *OS/2 Programming Reference* for more information on threads.

Ending the Extended Processing

There are two ways to end the extended processing.

- The thread completes its task.
- The user cancels the action by selecting a choice on the displayed panel.

The DM application can determine how the DISPLAY service ended by interrogating the DISPLAY service reason code value.

Completed Task

If the thread completes its task, the thread must force the Dialog Manager to return from the related DISPLAY service. Do this by using the FORCEXIT service.

User Cancels Action

If the user cancels the action by selecting a choice on the panel, the application should either stop the thread or ignore any results from the processing of that thread. In the Dialog Manager thread, you should then use the FORCEXIT service with the CLEAR parameter to clear any pending FORCEXIT request. Do this because the non-Dialog Manager thread may have requested the FORCEXIT service *after* the user cancelled the operation. The FORCEXIT service with the CLEAR parameter removes the duplicate request (if any) to exit the current DISPLAY service.

Example Using FORCEXIT Service

The following example shows the code for a DM application that creates a separate thread to search a file. In this case, the user is allowed to cancel the operation at any time.

Note: The DM communication area (in this example, *WorkingThreadDMComm1*) is assumed to be in storage that is shared by both threads. This DM communication area is a copy of the main routine's DM communication area. Using a copy avoids overwriting the return and reason codes of the main routine's DM communication area when it calls the FORCEXIT service.

```
/*********************************************************************/
/* This sample code uses the FORCEXIT service in DM.                 */
/*                                                                   */
/* It creates another thread and executes WorkingThreadRoutine().    */
/* When the second thread is finished processing, it signals the     */
/* first thread by requesting FORCEXIT.                              */
/*********************************************************************/

/*********************************************************************/
/* Include files.                                                    */
/*********************************************************************/

#define INCL_DOS
#include "os2.h"                        /* include OS/2 functions    */

#include "string.h"                     /* include string C functions  */
#include "stdlib.h"                     /* include standard C functions*/
#include "memory.h"                     /* include memory C functions  */

#include "ispcast.h"                    /* include DM definitions    */
#include "isperror.h"                   /* include DM error constants  */

/*********************************************************************/
/* Constants.                                                        */
/*********************************************************************/

#define WORKING_THREAD_STACKSIZE 20480

#define FILE_ATTRIBUTE        0x0001
#define ACTION_IF_EXISTS      0x0011
#define OPEN_MODE             0x0010

/*********************************************************************/
/* Function declarations.                                            */
/*********************************************************************/

extern PFNTHREAD _loadds WorkingThreadRoutine(VOID);
```

```
/*********************************************************************/
/* Program variables for DM.                                         */
/*                                                                   */
/* Note these must be global since they are referenced by the main()  */
/* routine and WorkingThreadRoutine().                               */
/*********************************************************************/

DMCOMMBLOCK WorkingThreadDMComm;     /* copy of the main() routine     */
                                     /*   local DM communication block */

BOOL StopWorking = FALSE;            /* Cancel = stop flag             */
DOSFSRSEM StopWorkingSem = {sizeof(DOSFSRSEM), 0, 0, 0, 0, 0};
                                     /* semaphore for controlling      */
                                     /*   access to StopWorking flag   */

BOOL CharFound;                      /* TRUE if $ character found      */

/*********************************************************************/
/* Begin main() routine.                                             */
/*********************************************************************/

extern void main()
{
  USHORT returnThreadId;               /* ID of the thread created     */
  SEL returnStackSel;                  /* selector of the stack        */
  USHORT dosRC;                        /* OS/2 return code             */
  CHAR *service;                       /* DM service to be performed   */
  BOOL dmOpenWorked;                   /* DMOPEN flag                  */
  DMCOMMBLOCK dmcomm;                  /* DM communications block      */

  /*******************************************************************/
  /* Perform DMOPEN.  This initializes the Dialog Manager.           */
  /*******************************************************************/

  service = "DMOPEN APPLID(FORC)";
  ISPCI(&dmcomm,(LONG)strlen(service),service);

  if (dmcomm.ReturnCode > 0)           /* Did the DMOPEN service work? */
  {
    dmOpenWorked = FALSE;              /* it failed, so exit.          */
  }
  else
  {                                    /* dmOpenWorked = TRUE          */
    dmOpenWorked = TRUE;

    /*****************************************************************/
    /* Copy the local DM communications block to that referenced by  */
    /* WorkingThreadRoutine().  The WorkingThreadRoutine() uses a    */
    /* copy of the main() routine's DM communications block so that  */
    /* the return/reason codes of the main() routine's DM            */
    /* communications block are not overwritten when it calls FORCEXIT.*/
    /*****************************************************************/

    memcpy(&WorkingThreadDMComm,&dmcomm,sizeof(DMCOMMBLOCK));
```

```
/*****************************************************************/
/* Tell the Dialog Manager where to find the panel libraries.    */
/*****************************************************************/

service = "LIBDEF LIBRARY LIBLIST(FORCESMP.DTL)";
ISPCI(&dmcomm,(LONG)strlen(service),service);

if (dmcomm.ReturnCode == 0)        /* Did the LIBDEF service work?*/
{
  /***************************************************************/
  /* Allocate space for a stack to be used by the DosCreateThread */
  /* call.  This calls returns the stack segment selector in the  */
  /* variable returnStackSel.                                     */
  /***************************************************************/

  dosRC = DosAllocSeg(WORKING_THREAD_STACKSIZE,&returnStackSel,0);

  if (!dosRC)                      /* Was the stack allocated?    */
  {
    /*************************************************************/
    /* Call the OS/2 function to create another thread.  The      */
    /* address of the routine to execute in another thread and    */
    /* the stack pointer are input parameters to this call.  The  */
    /* thread ID of the newly created thread is returned in       */
    /* returnThreadId.                                            */
    /*************************************************************/

    dosRC = DosCreateThread((PFNTHREAD) WorkingThreadRoutine,
        &returnThreadId,
        MAKEP(returnStackSel,WORKING_THREAD_STACKSIZE-1));

    if (!dosRC)                    /* Was thread created?         */
    {
      /***********************************************************/
      /* Display a panel notifying the user that calculations     */
      /* are being processed, and to wait or cancel the function. */
      /***********************************************************/

      service = "DISPLAY PANEL(WAITPAN)";
      ISPCI(&dmcomm,(LONG)strlen(service),service);

      /***********************************************************/
      /* Test to see if the user cancelled the function by        */
      /* pressing Esc or if the other thread finished its         */
      /* processing and requested FORCEXIT.                       */
      /***********************************************************/

      if (dmcomm.ReasonCode != DMERR_FORCEXIT_INVOKED)
      {
        /*********************************************************/
        /* The user pressed Esc.  Signal working thread to stop.  */
        /* Perform a FORCEXIT CLEAR to clear pending FORCEXIT     */
        /* calls.                                                 */
        /*********************************************************/
```

```
            /*********************************************************/
            /* The semaphores are used to ensure the StopWorking flag */
            /* is not being tested and set at the same time.          */
            /*********************************************************/

            DosFSRamSemRequest(&StopWorkingSem,SEM_INDEFINITE_WAIT);
            StopWorking = TRUE;
            DosFSRamSemClear(&StopWorkingSem);

            service = "FORCEXIT CLEAR";
            ISPCI(&dmcomm,(LONG)strlen(service),service);
          }                              /* ForceExit invoked          */
        }                                /* Was thread created?        */
      }                                  /* Was the stack allocated?   */
    }                                    /* Did Libdef work?           */
  }                                      /* dmOpenWorked = TRUE        */

  /*********************************************************************/
  /* Notify the Dialog Manager that the application is ending.         */
  /*********************************************************************/

  if (dmOpenWorked)
  {
    service = "DMCLOSE";
    ISPCI(&dmcomm,(LONG)strlen(service),service);
  }
}  /* main */

/*********************************************************************/
/* WorkingThreadRoutine:  Routine to run in another thread.          */
/*                                                                   */
/* This procedure opens a file and does some searching.  When the    */
/* searching is complete, a FORCEXIT service call is performed to    */
/* notify the main DM thread that processing is complete.            */
/*                                                                   */
/* The '_loadds' keyword below tells the compiler that DS needs to be */
/* set before entering this routine.  This is necessary because this  */
/* thread is not on the same thread as main() and therefore has its   */
/* own stack and thus DS!=SS.                                        */
/*                                                                   */
/* See "Creating Custom Memory Models" in the C/2 Compile, Link, and  */
/* Run documentation for more details.                               */
/*********************************************************************/

extern PFNTHREAD _loadds WorkingThreadRoutine()
{
  HFILE fileHandle;                  /* file handle                */
  USHORT actionTaken;                /* DosOpen action taken       */
  USHORT dosRC;                      /* OS/2 return code           */
  CHAR rbuff[1];                     /* read buffer                */
  USHORT bytesRead;                  /* number of bytes read       */
  CHAR *service;                     /* DM service to be performed */
  BOOL fileOpened;                   /* TRUE if file opened        */
```

```
/*********************************************************************/
/* Open the file to be searched.                                     */
/*********************************************************************/

dosRC = DosOpen("MYNAME.TXT",&fileHandle,&actionTaken,0L,
    FILE_ATTRIBUTE,ACTION_IF_EXISTS,OPEN_MODE,0L);
fileOpened = !dosRC;

/*********************************************************************/
/* If the file was opened successfully then read from it.            */
/*********************************************************************/

if (fileOpened)
{
  CharFound = FALSE;

  dosRC = DosRead(fileHandle,rbuff,sizeof(rbuff),&bytesRead);

  while (!CharFound && !dosRC && bytesRead==sizeof(rbuff) &&
      !StopWorking)
  {
    /*****************************************************************/
    /* Look for the character '$' in the file.  Set the flag if it   */
    /* is found.                                                     */
    /*****************************************************************/

    if (rbuff[0]=='$')
      CharFound = TRUE;
    else
      dosRC = DosRead(fileHandle,rbuff,sizeof(rbuff),&bytesRead);
  }
}

/*********************************************************************/
/* Close the file.                                                   */
/*********************************************************************/

if (fileOpened)
  DosClose(fileHandle);
```

```
  /*********************************************************************/
  /* Signal the main thread that the work is complete.                 */
  /*                                                                   */
  /* A fast safe ram semaphore is used to control access to the        */
  /* StopWorking flag between both threads.  It is used to protect     */
  /* the situation where one thread is testing it while the other      */
  /* thread is setting it, thus causing unpredictable results.         */
  /*********************************************************************/

  DosFSRamSemRequest(&StopWorkingSem,SEM_INDEFINITE_WAIT);

  if (!StopWorking)
  {
    service = "FORCEXIT";
    ISPCI(&WorkingThreadDMComm,(LONG)strlen(service),service);
  }

  DosFSRamSemClear(&StopWorkingSem);

} /* WorkingThreadRoutine */
```

The following panel definition is used to create a panel that tells the user to wait while the DM application completes the processing in the separate thread.

```
<!--*********************************************************************-->
<!--  GML file for the panel to tell the user to wait while          -->
<!--  calculations are being conducted.                              -->
<!--*********************************************************************-->
<!doctype dm system>

<!--*********************************************************************-->
<!--  The panel definition for the WAIT panel.                       -->
<!--*********************************************************************-->

<panel name=waitpan depth=10 width=35 >WAIT
<botinst>... Please wait. Press Cancel to interrupt.
</panel>
```

User Controls and User Exits

This section describes user controls and user exits, which are facilities that the Dialog Manager provides so that you can customize or extend your DM applications. The terms are defined and the similarities and differences between the two are discussed. The last section contains message syntax for user controls and for user exits.

What Is A User Control?

The Dialog Manager provides a way for you to use Presentation Manager to create your own panel elements within a Dialog Manager panel. These panel elements are called *user controls*, and they are defined on a panel using the UC tag. You must write code in order to support a user control, unlike all of the other panel elements provided by Dialog Manager.

User controls can perform a fairly simple function or they can be quite complex. For example, a simple user control may draw a basic graphics picture. A more complex user control could track the mouse, handle input, and draw complicated pictures, such as a graphics editor might do.

To write the code that supports a user control, you must understand Presentation Manager architecture and how to write window procedures. Refer to the *Presentation Manager Programming Reference* for more information.

What Is a User Exit?

The Dialog Manager provides a facility that allows you to extend certain Dialog Manager functions in order to better suit your application needs. This facility is called a user exit, and user exits allow you to get control during panel processing, rather than after the DISPLAY service has returned. There are five types of user exits provided by Dialog Manager:

Action Exit
Dialog Manager provides the RUN, SETVAR, and TOGVAR actions, but if you need to provide another action, you can do so using the action exit. All actions coded under a CHOICE or PDC tag are performed when the selection field choice or pull-down choice is selected. An action exit is coded using the CLASS attribute on the ACTION tag.

Check Exit
Allows you to perform your own checks on the data that is entered on a panel. A check exit is coded using the CLASS attribute on the CHECKI tag.

Command Action Exit
Allows you to perform a command action different than ALIAS, PASSTHRU, or SETVERB when a command is processed. A command action exit is coded using the CLASS attribute on the CMDACT tag.

Translate Exit
Allows you to perform your own translation on the data that is displayed on a panel or entered by the user. A translate exit is coded using the CLASS attribute on the XLATL tag.

Variable Access Exit
Allows you to use a variable access method different from variable pools. This exit is coded using the CLASS attribute on the VARLIST tag.

To write the code that supports a user exit, you must understand Presentation Manager architecture and how to write window procedures. Refer to the *Presentation Manager Programming Reference* for more information.

Basic Concepts—User Controls and User Exits

This section describes concepts needed to understand both user controls and user exits. The individual sections that describe user controls and user exits will reference this section for details of concepts common to both functions.

Registering a Window Class

All user control and user exit classes must be registered with Presentation Manager using the WinRegisterClass function. Use the following rules when registering your class:

- Combine (bitwise OR) the class style constant provided by Dialog Manager with your class style constant. Use this value as the value of the flStyle parameter on the WinRegisterClass() call.
- Add the extra bytes constant provided by Dialog Manager to your extra bytes constant. Use this value as the value of the cbWindowData parameter on the WinRegisterClass() call.

Note: For command action user exits, the following rules apply:

  - Use only the class style constant provided by Dialog Manager as the value of the flStyle parameter on the WinRegisterClass() call.
  - Use only the extra bytes constant provided by Dialog Manager as the value of the EXTRA parameter on the WinRegisterClass() call.

- The window procedure address passed as a parameter on WinRegisterClass() must be the window procedure that handles the messages sent to the user control or user exit.
- A user control class can be registered as a public class. Refer to the *Presentation Manager Programming Reference* for the rules regarding registering a public class. Note that the registration will have to occur when the PM shell is initialized. This requires changing the OS2.INI file.

The following example shows how the application should register a user control class.

```
WinRegisterClass((HAB)NULL,       /*  Anchor block handle          */
  "MyWindow",                     /*  User control class name      */
  MyWindowProc,                   /*  User control window procedure */
  CCTL_UC_CLASS_STYLE,            /*  User control window style    */
  CCTL_UC_EXTRA_BYTES);           /*  Extra bytes in user control  */
```

User exit classes are registered the same way, except the constants for class style and extra bytes are as follows:

User Exit	Class Style Constant	Extra Bytes Constant
Action	CCTL_UA_CLASS_STYLE	CCTL_UA_EXTRA_BYTES
Check	CCTL_UK_CLASS_STYLE	CCTL_UK_EXTRA_BYTES
Command Action	CCTL_UM_CLASS_STYLE	CCTL_UM_EXTRA_BYTES
Translate	CCTL_UX_CLASS_STYLE	CCTL_UX_EXTRA_BYTES
Variable Access	CCTL_UV_CLASS_STYLE	CCTL_UV_EXTRA_BYTES

Subclassing a Dialog Manager Class

All user controls and user exits must subclass the appropriate Dialog Manager class so that Dialog Manager default behavior is provided. In order to subclass the appropriate Dialog Manager user control or user exit class, your window procedure calls the window procedure for the Dialog Manager user control or user exit instead of calling WinDefWindowProc(). Your window procedure should call the Dialog Manager user control or user exit if it does not process a message it received. This ensures that all messages sent to the user control or user exit are handled properly.

The following example illustrates how a user control window procedure subclasses the Dialog Manager user control class:

```
/*********************************************************************/
/* Begin switch statement to check the incoming message.             */
/*********************************************************************/

switch (msg)
{
  case CM_xxx:
:
    break;

  case WM_xxx:
:
    break;

  case WM_xxx:
:
    break;
:
  default:
    /*****************************************************************/
    /* If the user control window procedure does not handle the      */
    /* incoming message, pass it on to the DM user control window    */
    /* procedure.  To do this, query the address of the DM user      */
    /* control window procedure.                                     */
    /*****************************************************************/

    WinQueryClassInfo((HAB)NULL,    /* Anchor block handle           */
      "DM_USER_CONTROL",            /* DM user control class name    */
      &classInfo);                  /* Address of class info struct */

    /*****************************************************************/
    /* Call the user control window procedure using the address      */
    /* returned by WinQueryClassInfo().                              */
    /*****************************************************************/

    mr = (*classInfo.pfnWindowProc)(hwnd, msg,
                                    mp1, mp2);
    break;
}
```

User exit window procedures subclass the same way, except the class name of the Dialog Manager user exit class is different. The Dialog Manager user exit class names are:

User Exit	Name of DM Class to Subclass
Action	DM_USER_ACTION
Check	DM_USER_CHECK
Command Action	DM_USER_CMD_ACTION
Translate	DM_USER_XLATE_LIST
Variable Access	DM_USER_VARIABLE

Initializing Message Parameters

When a user control or user exit sends a message to Dialog Manager, it must initialize all fields of the message parameter. In particular, it must initialize the MessageParmSize field to the number of bytes in the message's parameter structure. See "Message Syntax" on page 12-21 for a description of each message parameter structure.

Allocating Instance Data

The Dialog Manager provides a mechanism for user controls and user exits to have their own instance data. Instance data for a user control or user exit may exist for each instance of the class. For every user control and user exit, the Dialog Manager reserves a pointer (type PVOID) that can be used by the user control or user exit to point to instance data. The following is a list of rules to follow when allocating instance data for your user control or user exit:

- Set the pointer to the instance data when the user control or user exit receives an initialization message, by doing the following:
  1. Allocate the amount of memory needed for the user control or user exit instance data, by sending the CM_MEM_ALLOC message to your window handle.
  2. Set the pointer to the instance data by sending the CM_SET_USER_INST_PTR message to your window handle. This keeps the pointer to the user control or user exit instance data until the user control or user exit is destroyed.
- To get the pointer to the user control or user exit instance data, send the CM_QUERY_USER_INST_PTR message to your window handle.

Note: The QWL_USER window ULONG is reserved for use by the Dialog Manager.

See "Message Syntax" on page 12-21 for the complete syntax of all user control and user exit messages.

Freeing Resources

Every user control or user exit receives the CM_FREE_CTL_RESOURCES message. When this message is received, the user control or user exit should relinquish all system resources, such as memory and file handles, that it owns.

User Controls—Application Responsibilities

When writing a user control, you must do the following:

- Register the window class of the user control using WinRegisterClass(). See "Basic Concepts—User Controls and User Exits" on page 12-9 for more information.
- The window procedure for the user control must be subclassed from the Dialog Manager user control class, whose registered class name is "DM_USER_CONTROL." See "Basic Concepts—User Controls and User Exits" on page 12-9 for more information.
- The user control must not issue Dialog Manager services.
- If your user control needs to perform the ENTER, CANCEL or EXIT actions, you should send a CM_END_DISPLAY message to its window handle. See "Message Syntax" on page 12-21 for complete syntax of the CM_END_DISPLAY message.

User Control Size

The Dialog Manager needs to know the size of your user control so that it can be allotted the correct amount of space on the panel, and be positioned correctly. There are two ways to let Dialog Manager know the size of your user control:

1. If you code the MINWIDTH, MAXWIDTH, MINDEPTH, and MAXDEPTH attributes on the UC tag, Dialog Manager automatically sets your user control to the size specified on these attributes. Dialog Manager takes care of the necessary unit conversion, since MINWIDTH, MAXWIDTH, MINDEPTH, and MAXDEPTH are specified in character units. Using this option, your user control does not have to have any special code to handle setting the size of the user control.
2. If your user control needs to set the minimum and maximum widths and depths at run time, you can specify to the Dialog Manager a minimum and maximum size so that your user control can take on a range of sizes. To set the minimum and maximum width and depth of your user control, send a CM_SET_MIN_MAX_SIZE message to the window handle of the user control. This message should be sent before the panel containing the user control is visible, such as upon receipt of the CM_INIT_USER_CONTROL message. See "Message Syntax" on page 12-21 for more information about the CM_SET_MIN_MAX_SIZE message.

User Control Help and Cursoring

If the HELP attribute is coded on the UC tag, Dialog Manager automatically provides help on the user control. Your user control does not have to provide special code to support help.

Like all other panel elements on which help can be provided, the user control must be cursorable, so the user can move the cursor to the user control and select help. The Dialog Manager automatically makes all user controls cursorable. That is, the user control is included in the tabbing sequence on the panel.

Most panel elements visually indicate that they contain the cursor. For example, selection field choices draw a dashed box around their choice text and pull-down choices display in inverse video when they contain the cursor. It is an application responsibility to visually indicate that the user control contains the cursor, and this should be done when the user control either receives or loses the input focus, as indicated by the WM_SETFOCUS message.

If you do not want your user control to be in the tabbing sequence on the panel, you have to tell Dialog Manager by sending the CM_NO_FOCUS_USER_CONTROL message to its window handle. This message tells the Dialog Manager that the user control is not cursorable. Remember, if you tell Dialog Manager that your user control is not cursorable, the user will not be able to get help on your user control.

User Control Tag Attributes

The user control can access the parameter data and control text specified on the UC tag at run time by sending a CM_GET_UC_PARM_AND_TEXT message to its window handle. The structure for this message contains two pointers: one for the parameter data and one for the control text. If the parameter data or the control text was not specified on the UC tag, their respective pointers will be null.

User Exits—Application Responsibilities

This section describes the responsibilities common to all user exits.

- Register the window class of the user exit using WinRegisterClass(). See "Basic Concepts—User Controls and User Exits" on page 12-9 for more information.
- The window procedure for the user exit must be subclassed from the appropriate Dialog Manager user exit class. See "Basic Concepts—User Controls and User Exits" on page 12-9 for more information.
- The user exit must respond to the appropriate message for the specific type of user exit.
- The user exit must not issue Dialog Manager services.

The application responsibilities pertaining to the specific types of user exits are described in the following list:

- Action Exit

  The user action exit is expected to process two messages: CM_INIT_USER_ACTION and CM_DO_ACTION.

  A pointer to the value of the PARM attribute specified on the ACTION tag is passed to the exit in the message structures for CM_INIT_USER_ACTION and CM_DO_ACTION. If the PARM attribute was not specified on the ACTION tag, this pointer will be null.

  The CM_INIT_USER_ACTION message is sent to the exit indicating that initialization (if any) is to be performed. The ReturnCode field of the CM_INIT_USER_ACTION structure should be set to zero if initialization was successful or to CMERR_INIT_USER_ACTION if an error was encountered.

  The CM_DO_ACTION message is sent to the exit when the action is to be performed. The ReturnCode field of the CM_DO_ACTION message structure should be set to zero if the action was successful or to CMERR_UA_DO_ACTION if an error was encountered.

- Check Exit

  The user check exit is expected to process two messages: CM_INIT_USER_CHECK and CM_DO_CHECK.

  Pointers to the value of the PARM1 and PARM2 attributes specified on the CHECKI tag are passed to the exit in the message structures for CM_INIT_USER_CHECK and CM_DO_CHECK. If the PARM1 and/or PARM2 attributes were not specified on the CHECKI tag, their respective pointers will be null.

  The CM_INIT_USER_CHECK message is sent to the exit indicating that initialization (if any) is to be performed. The ReturnCode field of the CM_INIT_USER_CHECK structure should be set to zero if initialization was successful or to CMERR_INIT_USER_CHECK if an error was encountered.

  The CM_DO_CHECK message is sent to the exit when the check is to be performed. The ReturnCode field of the CM_DO_CHECK message structure should be set to zero if the action was successful or to CMERR_UK_DO_CHECK if an error was encountered.

- Command Action Exit

The user command action exit is expected to process two messages: CM_INIT_USER_CMD_ACTION and CM_DO_CMD_ACTION.

A pointer to the value of the PARM attribute specified on the CMDACT tag is passed to the exit in the message structures for CM_INIT_USER_CMD_ACTION and CM_DO_CMD_ACTION. If the PARM attribute was not specified on the CMDACT tag, this pointer will be null.

The CM_INIT_USER_CMD_ACTION message is sent to the exit indicating that initialization (if any) is to be performed. The ReturnCode field of the CM_INIT_USER_CMD_ACTION structure should be set to zero if initialization was successful or to CMERR_INIT_USER_CMD_ACTION if an error was encountered.

The CM_DO_CMD_ACTION message is sent to the exit when the action is to be processed. The ReturnCode field of the CM_DO_CMD_ACTION message structure should be set to zero if the action was successful or to CMERR_UM_DO_CMD_ACTION if an error was encountered.

- Translate Exit

  The user translate exit is expected to process two messages: CM_INIT_USER_XLATE and CM_DO_XLATE.

  A pointer to the value of the PARM attribute specified on the XLATL tag is passed to the exit in the message structures for CM_INIT_USER_XLATE and CM_DO_XLATE. If the PARM attribute was not specified on the XLATL tag, this pointer will be null.

  The CM_INIT_USER_XLATE message is sent to the exit indicating that initialization (if any) is to be performed. The ReturnCode field of the CM_INIT_USER_XLATE structure should be set to zero if initialization was successful or to CMERR_INIT_USER_XLATE if an error was encountered.

  The CM_DO_XLATE message is sent to the exit when the translation is to be performed. The ReturnCode field of the CM_DO_XLATE message structure should be set to zero if the action was successful or to CMERR_UX_DO_XLATE if an error was encountered.

  The message structure contains a constant that indicates the direction of the translation. The constant CCTL_INPUT indicates that the direction is input (post-display) and the constant CCTL_OUTPUT indicates that the direction is output (pre-display).

  The exit should use the null-terminated string pointed to by XlateBufferPointer of the message structure as input to the translation operation. Once the translation is complete, the translated value should be placed back into this buffer. The length of the buffer is contained in XlateBufferLen of the message structure.

- Variable Access Exit

  The Dialog Manager uses variable pools to store variable data. These pools are the source of variable data for panel, message and command table controls. A Presentation Manager control, called the Dialog Manager variable control, is responsible for moving data between the variable pools and the controls that use the data. The following diagram illustrates the relationship between the Dialog Manager variable control and other controls that use variable data.

GML Source:

```
<!DOCTYPE DM SYSTEM>

 <VARCLASS NAME=c1 TYPE='CHAR 1'>

 <VARLIST>
  <VARDCL NAME=v1 VARCLASS=c1>
  <VARDCL NAME=v2 VARCLASS=c1>
 </VARLIST>

<PANEL NAME=VARPAN>
 <DTAFLD NAME=d1 DATAVAR=v1>
 <DTAFLD NAME=d2 DATAVAR=v2>
</PANEL>
```

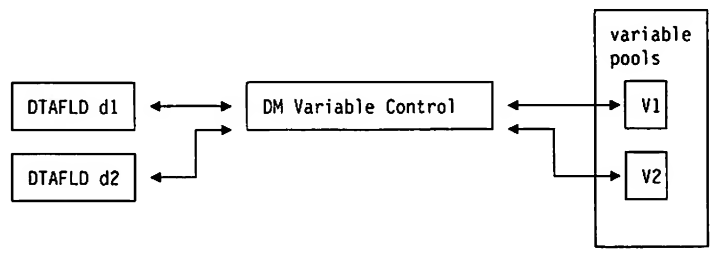


Figure 12-1. Relationship Between Variable Control and Other Controls

The CLASS attribute of VARLIST tag allows you to specify a control different from the Dialog Manager variable control to manage this flow of data between the data repository and the panel. For example, you could create a Presentation Manager window procedure called RelDBWndProc of the class RelDBClass that gets and puts variables from a relational database. By specifying CLASS=RelDBClass on the VARLIST tag, you tell the Dialog Manager that RelDBWndProc is to be used in place of the variable pools window procedure when accessing variables declared within that VARLIST. The following diagram illustrates the relationship between panel controls and variable controls when such a variable access user exit is used in conjunction with the Dialog Manager variable control.

GML Source:

```
<!DOCTYPE DM SYSTEM>

<VARCLASS NAME=c1 TYPE='CHAR 1'>

 <VARLIST>
  <VARDCL NAME=v1 VARCLASS=c1>
 </VARLIST>

 <VARLIST CLASS=RelDBClass>
  <VARDCL NAME=v2 VARCLASS=c1>
 </VARLIST>

<PANEL NAME=VARPAN>
 <DTAFLD NAME=d1 DATAVAR=v1>
 <DTAFLD NAME=d2 DATAVAR=v2>
</PANEL>
```

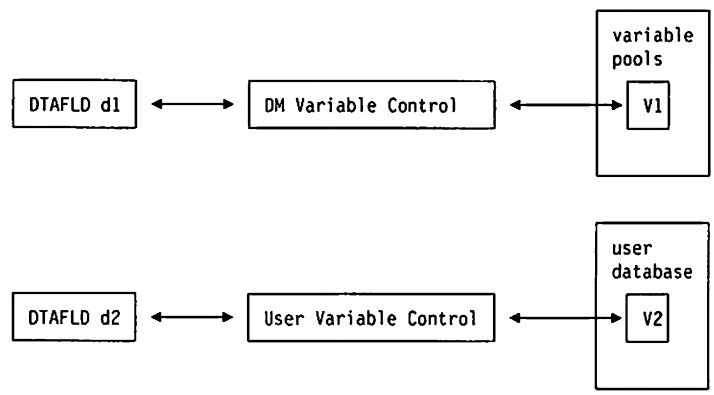


Figure 12-2. Relationship Between a User Variable Control and Other Controls

The variable access user exit must respond to Dialog Manager messages that query and set variable data. These messages are:

- CM_QUERY_VARIABLE_VALUE
- CM_SET_VARIABLE_VALUE

Complete syntax for these messages is in "Message Syntax" on page 12-21.

Example of a User Control

```
/*********************************************************************/
/* This is the window procedure for a sample user control.  This user */
/* control demonstrates how you can display simple graphics on a      */
/* Dialog Manager panel.  It draws a filled rectangle that changes    */
/* color between blue and green when the user clicks on it.  This     */
/* example also shows how to allocate instance data by keeping the    */
/* current color of the user control in the instance data.            */
/*********************************************************************/

#define INCL_WIN
#include "OS2.H"                    /* OS/2 global constants          */
#include "ISPUSER.H"                /* DM constants and typdefs for   */
                                    /* user controls and user exits   */

MRESULT APIENTRY UCWindowProc(HWND hwnd, USHORT msg,
                              MPARAM mp1, MPARAM mp2);

/*********************************************************************/
/* User control instance data                                        */
/*********************************************************************/

typedef struct
{
  LONG   UCColor;                   /* Color of user control          */
} UCInstanceData, far *UCInstanceDataPtr;

/*********************************************************************/
/* User control window procedure                                     */
/*********************************************************************/

MRESULT APIENTRY UCWindowProc(hwnd, msg, mp1, mp2)
HWND   hwnd;
USHORT msg;
MPARAM mp1;
MPARAM mp2;
{

  HPS          hps;                 /* Handle to presentation space   */
  RECTL        rectl;               /* User control window rectangle  */
  MRESULT      mr;                  /* Return value of winproc        */
  CLASSINFO    classInfo;           /* Class information structure    */
  UCInstanceDataPtr
               instanceDataPtr;     /* Pointer to user instance data  */
  cmMemAllocParm
               memAllocParm;        /* Message structure              */
  cmSetUserInstPtrParm
               setUserInstPtrParm;  /* Message structure              */
  cmQueryUserInstPtrParm
               queryUserInstPtrParm;/* Message structure              */

  mr = NULL;

  switch (msg)
  {
```

```
case CM_INIT_USER_CONTROL:

  /*************************************************************/
  /* Allocate memory for the user control instance data.       */
  /*************************************************************/

  memAllocParm.MessageParmSize = sizeof(cmMemAllocParm);
  memAllocParm.ByteCount = sizeof(UCInstanceData);
  instanceDataPtr =
     WinSendMsg(hwnd, CM_MEM_ALLOC, &memAllocParm, NULL);


  if (!memAllocParm.ReturnCode)
  {
    /*************************************************************/
    /* Set the pointer using the CM_SET_USER_INST_PTR message. */
    /* Initialize the field in the instance data that contains   */
    /* the default of the user control.                          */
    /*************************************************************/

    setUserInstPtrParm.MessageParmSize = sizeof(cmSetUserInstPtrParm);
    setUserInstPtrParm.UserInstancePtr = instanceDataPtr;
    WinSendMsg(hwnd, CM_SET_USER_INST_PTR, &setUserInstPtrParm,
        NULL);

    instanceDataPtr->UCColor = CLR_BLUE;

  }
  else
  {
    /*************************************************************/
    /* If there was an error allocating memory, return the error */
    /* to Dialog Manager.                                        */
    /*************************************************************/

    ((cmInitUserControlParmPtr)mp1)->ReturnCode =
        memAllocParm.ReturnCode;
  }
  break;
```

```
    case WM_PAINT:

      /*****************************************************************/
      /* Query the pointer to the user control instance data, and get */
      /* the color of the user control.  Then paint user control by   */
      /* filling a rectangle.                                          */
      /*****************************************************************/

      queryUserInstPtrParm.MessageParmSize =
          sizeof(cmQueryUserInstPtrParm);
      instanceDataPtr = WinSendMsg(hwnd, CM_QUERY_USER_INST_PTR,
          &queryUserInstPtrParm, NULL);

      hps = WinBeginPaint(hwnd, NULL, &rectl);
      WinFillRect(hps, &rectl, instanceDataPtr->UCColor);
      WinEndPaint(hps);
      break;

    case WM_BUTTON1DOWN:

      /*****************************************************************/
      /* Toggle the color of the user control, and force the user     */
      /* control to repaint.                                           */
      /*****************************************************************/

      queryUserInstPtrParm.MessageParmSize = sizeof(cmQueryUserInstPtrParm);
      instanceDataPtr = WinSendMsg(hwnd, CM_QUERY_USER_INST_PTR,
         &queryUserInstPtrParm, NULL);

      if (instanceDataPtr->UCColor == CLR_BLUE)
        instanceDataPtr->UCColor = CLR_GREEN;
      else
        instanceDataPtr->UCColor = CLR_BLUE;

      WinInvalidateRect(hwnd, NULL, FALSE);
      break;

    default:

      /*****************************************************************/
      /* Query the DM User Control class for its class information    */
      /* structure - this contains the window procedure address.      */
      /* Use the procedure address to pass the message on to the DM   */
      /* User Control.                                                 */
      /*****************************************************************/

      WinQueryClassInfo((HAB)NULL, "DM_USER_CONTROL",
          &classInfo);
      mr =  (*classInfo.pfnWindowProc)(hwnd, msg, mp1, mp2);
      break;
  }

  return mr;
}
```

Include Files

You must include the following include file provided by Dialog Manager in the user exit or user control window procedure:

File	Description
ISPUSER.H	Constants and typedefs used by Dialog Manager user controls and user exits.

Message Syntax

This section describes the messages that user controls and user exits receive and send. Messages that are common among all user exits and user controls are described first. Messages that are specific to the user control and user exits are described in later sections.

Messages Common to User Controls and User Exits

This section describes messages that are common to user controls and user exits, and includes messages that are received by and sent from user controls and user exits. If user controls or user exits send private messages, these messages should be greater than or equal to the DM_USER constant.

Messages received

CM_FREE_CTL_RESOURCES

This message is sent to the user control or user exit to notify it to release any system resources such as memory or file handles.

mp1

Address of structure of type cmFreeCtlResourcesParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

mp2

NULL

Return

NULL

CM_UPDATE_CONTROL

This message is sent to the user control or user exit to notify it that the panel is being displayed or redisplayed.

mp1

Address of structure of type cmUpdateControlParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

SetInitialState (BOOL)

When this value is TRUE, the user control or user exit should set itself to its initial state.

mp2

NULL

Return

NULL

Messages sent

CM_MEM_ALLOC

The user control or user exit should send this message to itself to allocate any memory that it requires.

mp1

Address of structure of type cmMemAllocParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

ByteCount (USHORT)

Number of bytes of memory to allocate.

mp2

NULL

Return

Address of memory allocated or NULL if an error occurred.

CM_MEM_FREE

The user control or user exit sends this message to itself to free any memory allocated with the CM_MEM_ALLOC message.

mp1

Address of structure of type cmMemFreeParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

MemoryPtr (PVOID)

Pointer to memory to free.

mp2

NULL

Return

NULL

CM_MEM_REALLOC

The user control or user exit sends this message to itself to reallocate memory that was previously allocated with the CM_MEM_ALLOC message. It initializes the newly allocated memory with the data from the existing allocation.

mp1

Address of structure of type cmMemReallocParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

NewByteCount (USHORT)
The new size (in bytes) of the allocated memory.

SourcePtr (PVOID)
Pointer to memory to be reallocated.

mp2
NULL

Return
Pointer to reallocated memory or NULL if an error occurred.

CM_QUERY_CTL_HDL
The user control or user exit sends this message to obtain the handle to a named panel element. This message must be sent to the frame control which is the owner of the user control or user exit.

mp1
Address of structure of type cmQueryCtlHdlParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

Name (PSZ)
Pointer to the name of the panel element being searched for.

mp2
NULL

Return
Handle to the panel element being searched for or NULL if the element could not be found.

CM_QUERY_USER_INST_PTR
The user control or user exit sends this message to itself to query its instance data pointer which was previously saved with CM_SET_USER_INST_PTR.

mp1
Address of structure of type cmQueryUserInstPtrParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

mp2
NULL

Return
Pointer to instance data for the user control or user exit.

CM_SET_USER_INST_PTR

The user control or user exit sends this message to itself to save a pointer to its specific instance data.

mp1

Address of structure of type cmSetUserInstPtrParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

UserInstancePtr (PVOID)

Pointer to user control or user exit specific instance data.

mp2

NULL

Return

NULL

User Control Messages

This section describes messages that are received by and sent from user controls.

Messages received

CM_INIT_USER_CONTROL

The user control is sent this message to notify it to initialize itself.

mp1

Address of structure of type cmInitUserControlParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

mp2

NULL

Return

NULL

CM_PANEL_ENTERED

The user control is sent this message to notify it that the panel has been entered by the user so that it may do any processing needed.

mp1

Address of structure of type cmPanelEnteredParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

mp2

NULL

Return

NULL

Messages sent

CM_END_DISPLAY

The user control sends this message to itself when it wants to end the display of the panel.

mp1

Address of structure of type cmEndDIsplayParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

DisplayReasonCode (ULONG)

Reason code to be given to the application for the display request. This may be a DM supplied error code, such as DMERR_PANEL_ENTERED, or a user defined error code. User defined error codes may be taken from the following ranges:

Error Level	Range
0	6600 - 6699
4	406600 - 406699
8	806600 - 806699
12	1206600 - 1206699
16	1606600 - 1606699
20	2006600 - 2006699

For descriptions of the error levels, see Chapter 17, "Dialog Manager Errors" on page 17-1.

MsgOnInvalidField (BOOL)

This flag indicates whether message panels should be displayed if a field on the panel is not valid. Setting this value to TRUE indicates that message panels should be displayed. Setting it to FALSE indicates that message panel should not be displayed.

mp2

NULL

Return

NULL

CM_GET_UC_PARM_AND_TEXT

The user control sends this message to itself when it needs access to the data specified on the PRMDATA attribute of the UC tag and to the tag content of the UC tag.

mp1

Address of structure of type cmGetUCParmAndTextParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

ReturnControlTextPtr (PSZ)
A pointer to the tag content of the UC tag. This value is NULL if none was specified.

ReturnControlPrmdataPtr (PSZ)
A pointer to the value specified on the PRMDATA attribute of the UC tag. This value is NULL if none was specified.

mp2
NULL

Return
NULL

CM_NO_FOCUS_USER_CONTROL
The user control sends this message to itself to notify Dialog Manager that it cannot receive the input focus.

mp1
Address of structure of type cmNoFocusUserControlParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

mp2
NULL

Return
NULL

CM_SET_MIN_MAX_SIZE
The user control sends this message to itself to set its minimum and maximum width and depth.

mp1
Address of structure of type cmSetMinMaxSizeParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

MinWidth (USHORT)
Minimum width, in pixels, of the control.

MaxWidth (USHORT)
Maximum width, in pixels, of the control.

MinDepth (USHORT)
Minimum depth, in pixels, of the control.

MaxDepth (USHORT)
Maximum depth, in pixels, of the control.

Note: The MinWidth, MaxWidth, MinDepth, and MaxDepth fields have a maximum value of 32,767.

mp2
NULL

Return
NULL

CM_YES_FOCUS_USER_CONTROL
The user control sends this message to itself to notify Dialog Manager that it can receive the input focus. Initially, all user controls may receive the focus without sending this message. This message allows the user control to reverse the effect of the CM_NO_FOCUS_USER_CONTROL.

mp1
Address of structure of type cmYesFocusUserControlParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

mp2
NULL

Return
NULL

User Action Exit Messages

This section describes the messages received by and sent from user action exits.

Messages received

CM_DO_ACTION
The user action exit is sent this message to notify it to perform its action.

mp1
Address of structure of type cmDoActionParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

ActionParmPtr (PSZ)
A pointer to the value of the PARM attribute specified on the ACTION tag. This value is NULL if none was specified.

mp2
NULL

Return
NULL

CM_INIT_USER_ACTION

The user action exit is sent this message to notify it to initialize itself.

mp1

Address of structure of type cmInitUserActionParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

ActionParmPtr (PSZ)

A pointer to the value of the PARM attribute specified on the ACTION tag. This value is NULL if none was specified.

mp2

NULL

Return

NULL

Messages sent

None.

User Check Exit Messages

This section describes the messages received by and sent from user check exits.

Messages received

CM_DO_CHECK

The user check exit is sent this message to notify it to perform its check.

mp1

Address of structure of type cmDoCheckParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

CheckValuePtr (PSZ)

A pointer to the value to be checked. This pointer and the storage it points to should never be modified.

CheckParm1Ptr (PSZ)

A pointer to the value of the PARM1 attribute specified on the CHECKI tag. This value is NULL if none was specified.

CheckParm2Ptr (PSZ)

A pointer to the value of the PARM2 attribute specified on the CHECKI tag. This value is NULL if none was specified.

mp2

NULL

Return

NULL

CM_INIT_USER_CHECK

The user check exit is sent this message to notify it to initialize itself.

mp1

Address of structure of type cmInitUserCheckParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

CheckParm1Ptr (PSZ)

A pointer to the value of the PARM1 attribute specified on the CHECKI tag. This value is NULL if none was specified.

CheckParm2Ptr (PSZ)

A pointer to the value of the PARM2 attribute specified on the CHECKI tag. This value is NULL if none was specified.

mp2

NULL

Return

NULL

Messages sent

None.

User Command Action Exit Messages

This section describes the messages that can be received by and sent from user command action exits.

Messages received

CM_DO_CMD_ACTION

The user command action exit is sent this message to notify it to perform its action.

mp1

Address of structure of type cmDoCmdActionParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

CmdParmPtr (PSZ)
A pointer to the parameters of the command if this command was entered in the command area. The string does not contain the command name. This value is NULL if no parameters were entered or if the command was not initiated from the command area.

CmdActionParmPtr (PSZ)
A pointer to the value of the PARM attribute specified on the CMDACT tag. This value is NULL if none was specified.

mp2
NULL

Return
NULL

CM_INIT_USER_CMD_ACTION
The user command action exit is sent this message to notify it to initialize itself.

mp1
Address of structure of type cmInitUserCmdActionParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

CmdActionParmPtr (PSZ)
A pointer to the value of the PARM attribute specified on the CMDACT tag. This value is NULL if none was specified.

mp2
NULL

Return
NULL

Messages sent

None.

User Translate Exit Messages

This section describes the messages that can be received by and sent from user translate exits.

Messages received

CM_DO_XLATE
The user translate exit is sent this message to notify it to perform its translation.

mp1
Address of structure of type cmDoXlateParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

Direction (USHORT)
Specifies the direction of the translation. Its value may be CCTL_INPUT, indicating input (post-display) translations are to be performed or it may be CCTL_OUTPUT, indicating output (pre-display) translations are to be performed.

XlateBufferLen (USHORT)
Maximum number of bytes a string being translated may ever become. It is taken from the DISPLEN attribute on the XLATL tag.

XlateBufferPtr (PSZ)
A pointer to the buffer containing the value to be translated. This buffer is also used to return the translated value. The XlateBufferLen field of this message parameter specifies the size of the buffer, in bytes.

XlateParmPtr (PSZ)
A pointer to the value of the PARM attribute specified on the XLATL tag. This value is NULL if not specified.

Continue (BOOL)
This tells the Dialog Manager whether to perform the translations and checks that follow this translation (if any).

mp2
NULL

Return
NULL

CM_INIT_USER_XLATE
The user translate exit is sent this message to notify it to initialize itself.

mp1
Address of structure of type cmInitUserXlateParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

XlateBufferLen (USHORT)
This is the maximum number of bytes a string being translated may ever become. It is taken from the DISPLEN attribute on the XLATL tag.

XlateParmPtr (PSZ)
A pointer to the value of the PARM attribute specified on the XLATL tag. This value is NULL if not specified.

mp2

NULL

Return

NULL

Messages sent

None.

User Variable Exit Messages

This section describes the messages that can be received by and sent from variable exits.

Messages received

CM_INIT_USER_VARIABLE

The user variable exit is sent this message to notify it to initialize itself.

mp1

Address of structure of type cmInitUserVariableParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

mp2

NULL

Return

NULL

CM_QUERY_VARIABLE_VALUE

The user variable exit is sent this message when the Dialog Manager needs to obtain the value of a variable.

mp1

Address of a structure of type cmQueryVariableValueParmPtr which contains:

ReturnCode (ULONG)

Return code of the message.

MessageParmSize (USHORT)

Number of bytes in the message structure.

VnamePtr (PSZ)

A pointer to the name of the variable being queried.

ReturnBufferPtr (PSZ)

A pointer to a user buffer in which the data will be returned.

ReturnBufferLength (USHORT)

Length of the ReturnBufferPtr.

Subscript (USHORT)
The subscript of the array variable being queried. This value is zero if the variable is scalar.

mp2
NULL

Return
Length of the variable data.

CM_SET_VARIABLE_VALUE
The user variable exit is sent this message when the Dialog Manager needs to set the value of a variable.

mp1
Address of a structure of type cmSetVariableValueParmPtr which contains:

ReturnCode (ULONG)
Return code of the message.

MessageParmSize (USHORT)
Number of bytes in the message structure.

VnamePtr (PSZ)
A pointer to the name of the variable being set.

BufferPtr (PSZ)
A pointer to the data to be assigned to the variable.

Subscript (USHORT)
The subscript of the array variable being queried. This value is zero if the variable is scalar.

mp2
NULL

Return
Number of bytes of data actually assigned to the variable.

Messages sent

None.

Chapter 13. Designing More Complex DM Applications

Every DM application must contain at least one DMOPEN service call and one DMCLOSE service call. A more complex DM application could call one or more subroutines that contain DMOPEN and DMCLOSE service calls. Each subroutine can also call a subroutine that performs a DMOPEN and DMCLOSE. Thus, a hierarchy of DMOPEN service calls can be built. The Dialog Manager does not limit the number of levels allowed in this hierarchy.

This chapter describes how to use multiple DMOPEN service calls to create more complex DM applications. It also provides an example that illustrates the use of multiple DMOPEN calls in a large application.

Multiple DMOPEN Service Calls

An application establishes a new dialog by issuing a DMOPEN service call without the INSTID parameter. You must create a new thread to establish a new dialog. Normally an application will contain only one DMOPEN. The Dialog Manager does, however, allow an application to contain more than one DMOPEN.

If multiple DMOPEN service calls are used, subsequent DMOPEN calls within an application must specify the INSTID parameter and be on the same thread if they are to belong to the same dialog (that is, to continue the current window hierarchy within the application's primary window). The value specified with the INSTID parameter is the *instance-id* assigned by the Dialog Manager to the application when the first DMOPEN is processed. The instance ID is found in the DM communication area associated with the first DMOPEN.

Also, an application can change the libraries containing the dialog elements to be used. This is done by specifying both the INSTID and APPLID parameters on nested DMOPEN service calls.

When an application uses multiple DMOPEN calls that share the same instance ID, all service requests relating to the windows (for example, DISPLAY and ADDPOP) operate in a single primary window and the pop-up windows associated with that primary window. However, there are differences. Each DMOPEN requires a unique DM communication area. Because all the DMOPEN calls are sharing the same dialog, each of these DM communication areas will contain the same instance ID. The following list summarizes the differences in the DMOPEN calls and the use of the DM communication areas associated with them.

- Each DMOPEN results in the creation of a new dialog variable pool.

  When you use a Dialog Manager service, the DM communication area you supply with that service will determine which dialog variable pool will be used by Dialog Manager.

  Note: When multiple programs share a single DM communication area, the stacked VDEFINE capability can be used to avoid conflicts with explicitly defined dialog variables.

  The use of the Procedures Language variable pool as the dialog variable pool prevents the Dialog Manager from providing a new dialog variable pool for each DMOPEN in a Procedures Language program. Therefore, multiple DMOPEN calls in a Procedures Language program would not be meaningful

except to access a new application library definition file by specifying a new APPLID.

- You must specify an application ID for the first DMOPEN in a dialog instance.

  Remember that the Dialog Manager uses application IDs to find the application library definition file, xxxxLIB.APP and the application command table, xxxxCMDS.DTL. Nested DMOPEN service calls within a single dialog inherit the current application ID if an application ID is not specified. Dialog elements (for example, panels, messages, and so on) can be shared by different dialogs by specifying the same application ID on multiple DMOPEN service calls.

- The first DMOPEN within an application must not specify the INSTID parameter.

  A program that is intended to be used as a building block (called by another program but continuing in the existing dialog) must accept an *instance-id* parameter and use it on its DMOPEN calls. If the same program is intended to run as a stand-alone program, it can check for the absence (or null value) of the *instance-id* parameter it accepts and perform its DMOPEN calls without the INSTID parameter if none was provided.

- If DMOPEN service calls are nested (within a dialog), they must be properly nested.

  This means that the DM communication area initialized by the most recent DMOPEN in a dialog must be used on all Dialog Manager service calls until it is closed using a DMCLOSE call. The Dialog Manager does not support nesting of DMOPENs with the alternating use of the DM communication areas on subsequent service calls.

- Support of multiple concurrent dialogs within a single application is dependent on the multitasking and windowing capabilities of the underlying system.

- Multiple dialogs with unique instance IDs (and running in separate primary windows) may be started by a single application if a separate process or thread is used for each dialog. Refer to the *OS/2 Operating System* book for details on this capability.

Multiple DMOPEN Example

The following example describes how to use multiple DMOPEN service calls in a DM application. In this scenario, a large application is coded in two separate parts. The parts are then merged together into a single application that has nested DMOPEN calls.

A software development company developing a large application, such as a document processor system, can use nested DMOPEN service calls. Suppose the document processor system is comprised of two components: an annotation component and a thesaurus component. Further, suppose that the document processor system is developed as a DM application. Two different software teams develop the annotation and thesaurus components. These components are also DM applications.

To prevent the software development teams from creating conflicting dialog variable names and panel names, each team is assigned a unique application ID (APPLID) and is required to issue a DMOPEN service call with this APPLID. The application ID for the annotation development group is ANNT. Its list of panel libraries is found in the ANNTLIB.APP file. Its list of application commands is found in ANNTCMDS.DTL. As you can see from Figure 13-1 on page 13-4, the

annotation function performs a DMOPEN using the application ID ANNT, a new DM communication area, and the instance ID created by the DMOPEN service call in the main routine.

Note: The annotation function performs a DMCLOSE when it has completed.

The application ID for the thesaurus processor component is called THES. Its list of panel libraries is found in the THESLIB.APP file. Its list of application commands is found in the THESCMDS.DTL file. As seen in the Figure 13-1 on page 13-4 the thesaurus function performs a DMOPEN using the application ID THES, a new DM communication area, and the instance ID created by the DMOPEN service call in the main routine.

The advantage of this design is that it allows both development groups to work independently. With each nested DMOPEN, a new function pool is allocated. Because each component can specify its own set of panel libraries in its application definition file, both applications may have panels with the same names, as long as the names of the libraries are unique.

The document processor example illustrates this advantage more specifically. Suppose the annotate component and thesaurus component both have a search capability. Both components have a panel named SEARCHIT; however, the contents of the panels are different. The annotate development group designed its SEARCHIT panel to query the user for a chapter or section heading to annotate. The thesaurus development group designed its SEARCHIT panel to allow the user to enter a word and obtain a list of synonyms. If the thesaurus component stores its panels in its own panel library, THESAURS.DTL and the annotate component uses ANNOTATE.DTL, any name conflict will be avoided. The THESAURS.DTL will be listed in the THESLIB.APP file and the ANNOTATE.DTL file will be listed in the ANNTLIB.APP file.

As seen in figure Figure 13-1 on page 13-4, the main component can call each subcomponent where a subcomponent also performs a DMOPEN service call. Thus, a DM hierarchy can be built.

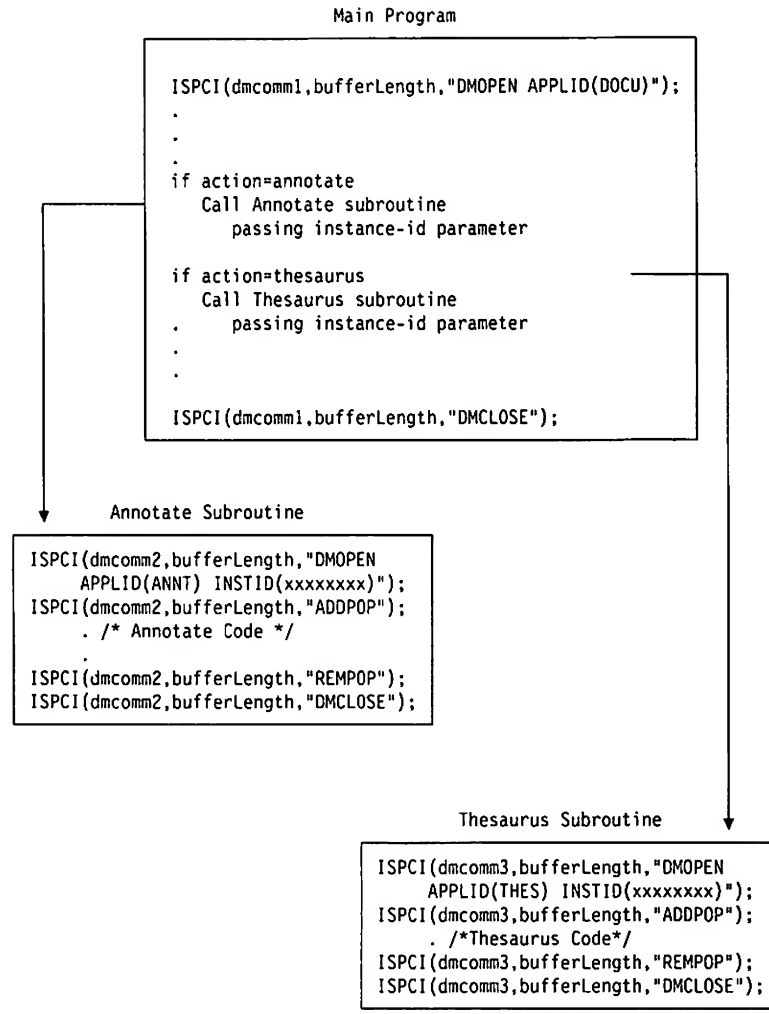


Figure 13-1. Document Processor DM Application

Note: The *xxxxxxxx* represents the *instance-id* returned in the *dmcomm1* area after the first DMOPEN.

Each subroutine performs an ADDPOP service at the beginning and a REMPOP service at the end of the routine. By performing the ADDPOP service, all panels displayed in this routine will be displayed in a pop-up window that is cascaded from the main panel. The main panel will still be displayed. This presentation interface complies with and is further described in the *SAA Common User Access: Advanced Interface Design Guide*, SC26-4582.

Part 2. Dialog Manager Reference

Chapter 14. Dialog Manager Services

This chapter is a reference chapter for the Dialog Manager services. The services are listed in alphabetical order for ease of use.

How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ▸▸— symbol indicates the beginning of a statement.

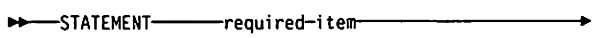
  The —▸ symbol indicates that the statement syntax is continued on the next line.

  The ▸— symbol indicates that a statement is continued from the previous line.

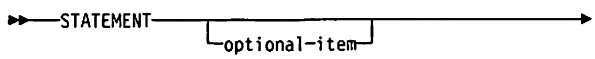
  The —▸◂ symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ▸— symbol and end with the —▸ symbol.

- Required items appear on the horizontal line (the main path).

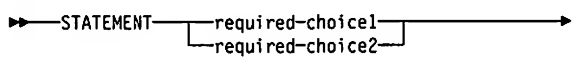


- Optional items appear below the main path.

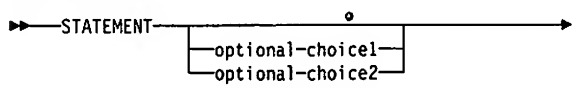


- If you can choose from two or more items, they appear vertically, in a stack.

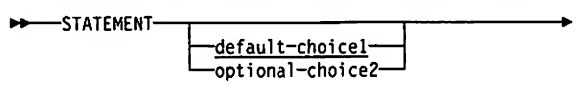
  If you *must* choose one of the items, one item of the stack appears on the main path.



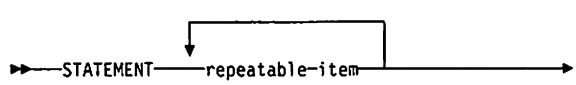
  If choosing one of the items is optional, the entire stack appears below the main path.



- Parameters that are underscored are default parameters. If you don't write it in the statement, you will get the same result as if you had actually written it.



- An arrow returning to the left above the item indicates an item that you can repeat. Required items appear on the main line and optional items appear below the main line.

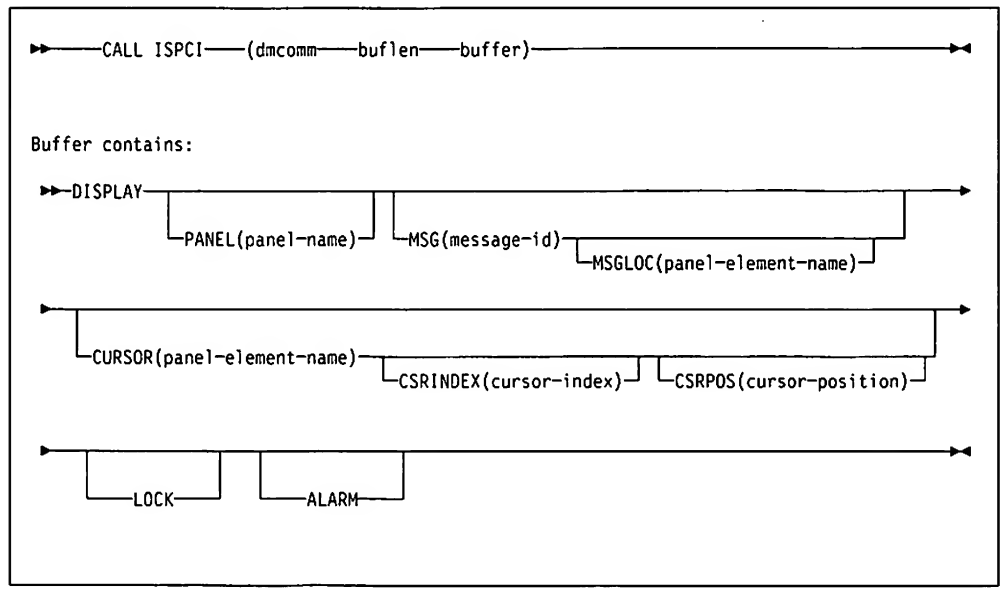


  A repeat arrow indicates that you can make more than one choice from the stacked items, or repeat a single item.

- Keywords appear in uppercase (for example, PARM1). However, they can be uppercase or lowercase when they are entered. They must be spelled exactly as shown. Variables and acceptable values appear in all lowercase letters (for example, parmx). They represent names or values that you supply. Keywords and keywords followed by keyword parameters (for example, MSG(message-id)) may be coded in any order.
- If punctuation marks, parentheses, arithmetic operators, or other symbols are shown, you must enter them as part of the syntax.

Syntax Example

The following diagram for the DISPLAY service illustrates the use of syntax diagrams.



Because CALL ISPCI is in all uppercase characters and appears on the main path, it is required and must be spelled exactly as shown.

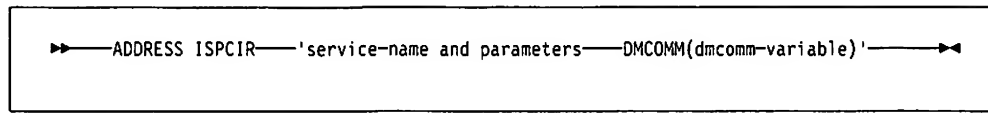
Next are dmcomm, buflen, and buffer, which are required but represent names or values that you supply.

When you code the buffer, DISPLAY is required and spelled exactly as shown.

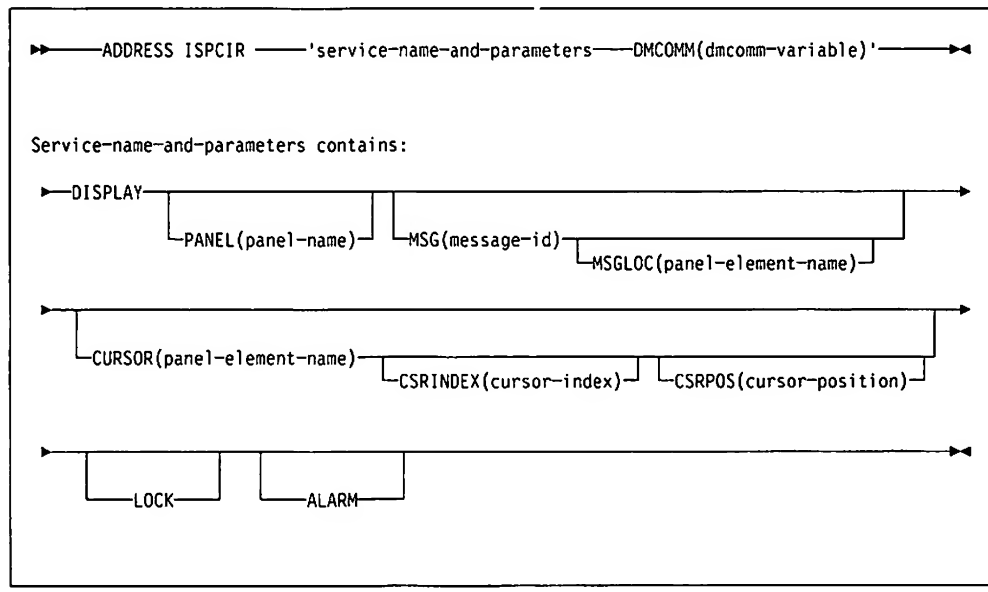
You can also optionally choose one or all of the following: PANEL, MSG, CURSOR, LOCK, and ALARM. Spell them exactly as shown; case does not matter. Supply a variable name or value for *panel-name*, *message-id*, and *panel-element-name*. MSGLOC is optional, but may be specified only if MSG is specified. CSRINDEX and CSRPOS are optional, and either one or both may be specified only if CURSOR is specified.

Procedures Language Syntax Example

The following illustrates the syntax of all Procedures Language Dialog Manager service calls.

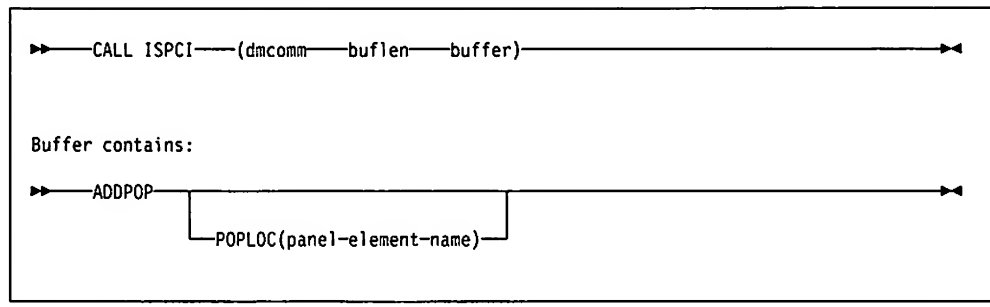


The following diagram for the DISPLAY service illustrates Procedures Language syntax.



ADDPOP

Use the ADDPOP service to display panels in pop-up windows.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

ADDPOP
The service name.

POPLOC(panel-element-name)
Used to position the pop-up window. If the DM application specifies this parameter, the Dialog Manager positions the pop-up relative to the named panel element in the underlying primary or pop-up window.

Only tags with the NAME attribute that result in a visual panel element can be used for pop-up window positioning. For example, the DTAFLD tag, which has the NAME attribute, can be used for pop-up window positioning because it results in a visual panel element. However, the VARDCL tag, which also has the NAME attribute, cannot be used for pop-up window positioning because it does not result in a visual panel element.

The reserved *panel-element-name*, CMDAREA, can be specified on the POPLOC parameter to position the pop-up window relative to the command area.

For additional information on how the NAME attribute is used, refer to the specific tag used to create the panel element.

If there is more than one element on the panel with the specified name, the Dialog Manager uses the first occurrence of that element to position the pop-up window.

If the POPLOC parameter is not specified, the Dialog Manager positions the window using offset positioning.

See "Window Positioning" on page 2-5 for a complete description of field-adjacent and offset pop-up window positioning.

Description

The ADDPOP service notifies the Dialog Manager that subsequent panel displays are to appear in a pop-up window.

Subsequent panel displays will be in the pop-up window created with the ADDPOP call, until a REMPOP or another ADDPOP service call is issued. Another ADDPOP call creates a separate (nested) pop-up window.

The number of separate pop-up windows that can be created using nested ADDPOP service calls is limited to six.

At least one DISPLAY service call, in order to create the primary window, must be issued prior to issuing any ADDPOP service calls.

Two successive ADDPOP service calls without an intervening DISPLAY service call that specifies a *panel-name* or REMPOP service call are not valid.

The REMPOP service call removes the pop-up window created by an ADDPOP call. For more information on the REMPOP service, see "REMPOP" on page 14-24.

Example

The following example displays panel *DME301* in a pop-up window overlaying panel *DMC301*, which is displayed in the primary window.

In C:

```
char    buffer[512];

strcpy(buffer, "DISPLAY PANEL(DMC301)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);

strcpy(buffer, "ADDPOP");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);

strcpy(buffer, "DISPLAY PANEL(DME301)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN           PIC S9(7) COMP VALUE 512.
 01 BUFFER           PIC X(512) VALUE SPACE.

    MOVE "DISPLAY PANEL(DMC301)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

    MOVE "ADDPOP" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

    MOVE "DISPLAY PANEL(DME301)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
      CHARACTER BUFFER*512

      INTEGER*4 BUFLEN

       BUFLEN = 21
       BUFFER = 'DISPLAY PANEL(DMC301)'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)

       BUFLEN = 6
       BUFFER = 'ADDPOP'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)

       BUFLEN = 21
       BUFFER = 'DISPLAY PANEL(DME301)'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm        dm_comm   <>
BFDMC301      db        "DISPLAY PANEL(DMC301)"
BFLDMC301     dd        21
BFADP         db        "ADDPOP"
BFLADP        dd        6
BFDME301      db        "DISPLAY PANEL(DME301)"
BFLDME301     dd        21

              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLDMC301
              push      ds
              push      ax
              lea       ax,BFDMC301
              push      ds
              push      ax
              CALL      ISPCI

              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLADP
              push      ds
              push      ax
              lea       ax,BFADP
              push      ds
              push      ax
              CALL      ISPCI

              lea       ax,dmcomm
              push      ds
              push      ax
              CALL      ISPCI
```

In Pascal:

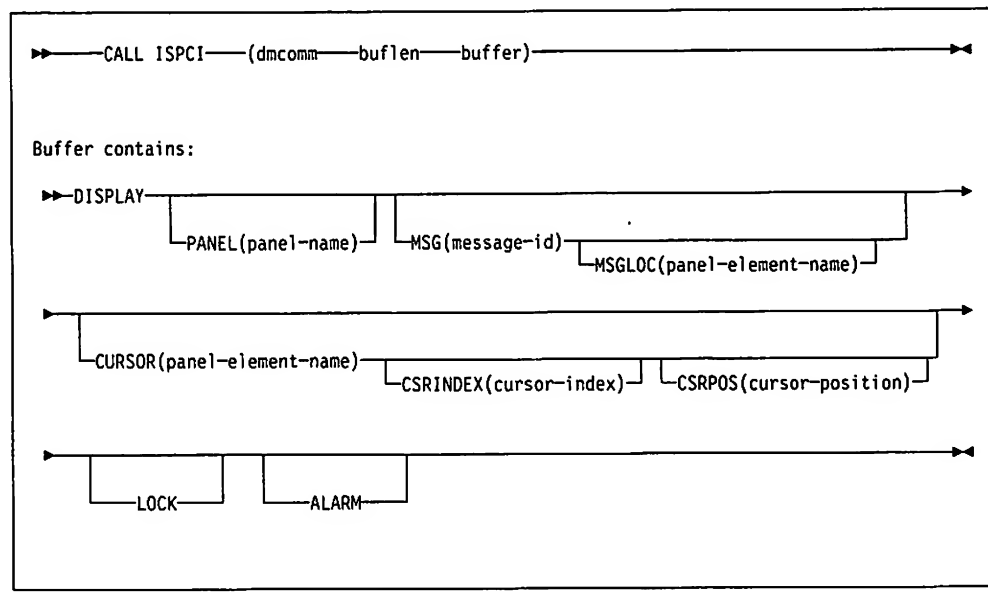
```
VAR [PUBLIC]
   BUFFER   :STRING(512);
   BUFLEN   :INTEGER4;


   COPYSTR('DISPLAY PANEL(DMC301)',BUFFER);
   BUFLEN := 21;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

   COPYSTR('ADDPOP',BUFFER);
   BUFLEN := 6;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

   COPYSTR('DISPLAY PANEL(DME301)',BUFFER);
   BUFLEN := 21;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

In Procedures Language:

```
ADDRESS ISPCIR 'DISPLAY PANEL(DMC301) DMCOMM(dmc)'

ADDRESS ISPCIR 'ADDPOP DMCOMM(dmc)'

ADDRESS ISPCIR 'DISPLAY PANEL(DME301) DMCOMM(dmc)'
```

DISPLAY

Use the DISPLAY service to display a panel.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

DISPLAY
The service name.

PANEL(panel-name)
The name of a panel. If omitted, the Dialog Manager reactivates the last panel displayed. The variable pool is not searched for new values, and the panel will be redisplayed with the previous display's values.

MSG(message-id)
The identification of a message to be displayed in a message pop-up associated with the panel.

MSGLOC(panel-element-name)
The *panel-element-name* the Dialog Manager uses to position the message pop-up containing the message specified by the MSG parameter.

Any visual panel element created with a tag that has the NAME attribute can be used for message pop-up positioning. For example, the DTAFLD tag, which has the NAME attribute, can be used for message pop-up positioning because it results in a visual panel element. However, the VARDCL tag, which also has the NAME attribute, cannot be used for message pop-up positioning because it does not result in a visual panel element.

Additionally, the reserved *panel-element-name* CMDAREA can be specified on the MSGLOC parameter to position the message pop-up relative to the command area.

For additional information on how the NAME attribute is used, refer to the *Dialog Tag Language Guide and Reference* to see the specific tag used to create the panel element.

If there is more than one element on the panel with the specified name, the Dialog Manager uses the first occurrence of that element to position the message pop-up.

If the MSGLOC parameter is not specified, and the message is to be displayed in a pop-up window, the Dialog Manager positions the window using offset positioning. See "Window Positioning" on page 2-5 for a complete description of pop-up window positioning.

CURSOR(panel-element-name)
The name of the panel element on which the Dialog Manager places the cursor. The *panel-element-name* you specify cannot be an output-only field. If there is more than one element on the panel with the specified name, the Dialog Manager uses the first occurrence of that element.

The cursor can also be placed on the command area, if one exists. The Dialog Manager reserves the name CMDAREA for this purpose.

If this parameter is not specified, the cursor is placed in the first field that allows input.

CSRINDEX(cursor-index)
An index indicating the row in the field specified on the CURSOR parameter in which to place the cursor. Index values less than one are not supported. The maximum value allowed is 1024.

This parameter applies only to list fields. The field specified must be an input list column.

If the DISPLAY call does not include this parameter, or its value is not in the field, the default value is 1.

CSRPOS(cursor-position)
The character position within the field where the Dialog Manager places the cursor. The field you specify for this parameter must be an input field.

The maximum value allowed is 1024.

If the DISPLAY call does not include this parameter, or its value is not in the field, the default value is 1.

LOCK
If this parameter is coded, input is not allowed, and control returns to the application once the panel is visible.

ALARM
Indicates that Dialog Manager should sound the alarm when the panel is displayed. The effect of this parameter is conditioned by the OS/2 system setting for the audible alarm.

Description

The DISPLAY service retrieves a panel definition, performs any pre-display processing required for the panel definition, initializes variable panel fields from the corresponding dialog variables, and displays the panel in the primary window or a pop-up window. The service can also display a message with the panel.

The user can type information in input fields or make selections as defined in the panel definition. Dialog Manager validates and processes user input according to the panel definition. When the user's interaction is complete, the Dialog Manager stores the contents of the input fields in dialog variables, and the DISPLAY service returns to the calling application program.

The LOCK parameter can be used to display a panel without soliciting user input. If LOCK is coded, the user cannot interact with the panel displayed, and control is returned to the application. See "Displaying Confirmation Messages" on page 5-3 for more information.

The DM application must give control to the Dialog Manager in a timely manner using a DISPLAY service call without specifying the LOCK option. The Dialog Manager can then yield control to the operating system. If the DM application must perform an extended operation, it should create a second thread and display an in-process panel. When the second thread has completed, it can notify the first thread using the FORCEXIT service. See "Forced Display Exit" on page 12-1 for more information.

The Dialog Manager processes the *panel-name* and *message-id* parameters as follows:

- If the DISPLAY call includes the *panel-name* parameter but does not include the *message-id* parameter, the Dialog Manager retrieves the panel and displays it.
- If the DISPLAY call includes both the *panel-name* and *message-id* parameters, the Dialog Manager retrieves the panel and displays it with the specified message in a pop-up window overlapping the current panel.
- If the DISPLAY call does not include the *panel-name* parameter but includes the *message-id* parameter, the Dialog Manager displays the specified message pop-up overlapping the current panel. Variables on the current panel are not refreshed from the dialog variable pool.
- If the DISPLAY call does not include the *panel-name* parameter or the *message-id* parameter, the Dialog Manager redisplays the current panel. Variables are not refreshed from the dialog variable pool.

If the *message-id* parameter is specified, the cursor placement information applies to the panel specified with the *panel-name* parameter. The Dialog Manager processes the cursor placement information before either the panel or message pop-up is displayed. If there is an error in positioning the cursor, an error is returned from the DISPLAY call, and neither the panel nor the message pop-up is displayed.

The end user interaction is considered complete, and the panel displayed is submitted for processing, when the user performs one of the following actions:

1. The user runs the ENTER action.
2. The user runs the CANCEL action.
3. The user runs the EXIT action.
4. The user runs a command with an action of PASSTHRU or SETVERB.

The PASSTHRU action may be implicitly run if the command table is defined with the NOMATCH=APPLCMD attribute of the CMD tag.

The following reason codes should be used to control program logic.

Reason Code	Description	Action Required
1700	The user selected Enter.	Perform application-specific ENTER processing.
1701	The user issued a command that is assigned to the PASSTHRU action.	Perform application-specific PASSTHRU processing.
1702	The user entered a command that is assigned to the SETVERB action.	Perform application-specific SETVERB processing.
401700	The user selected Esc=Cancel.	Perform application-specific CANCEL processing.
801700	The user selected Exit.	Perform application-specific EXIT processing.
801705	The FORCEXIT service was called.	The thread performing extended processing has completed. Continue application logic.

Example

In the following example, the Dialog Manager displays panel *DMM001* and places the cursor on the *name* field.

In C:

```
char    buffer[512];

strcpy(buffer, "DISPLAY PANEL(DMM001) CURSOR(name)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN          PIC S9(7) COMP VALUE 512.
 01 BUFFER          PIC X(512) VALUE SPACE.

    MOVE "DISPLAY PANEL(DMM001) CURSOR(NAME)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
      CHARACTER BUFFER*512

      INTEGER*4 BUFLEN

       BUFLEN = 34
       BUFFER = 'DISPLAY PANEL(DMM001) CURSOR(NAME)'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm        dm_comm   <>
BFDMM001      db        "DISPLAY PANEL(DMM001) CURSOR(NAME)"
BFLDMM001     dd        34

              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLDMM001
              push      ds
              push      ax
              lea       ax,BFDMM001
              push      ds
              push      ax
              CALL      ISPCI
```

In Pascal:

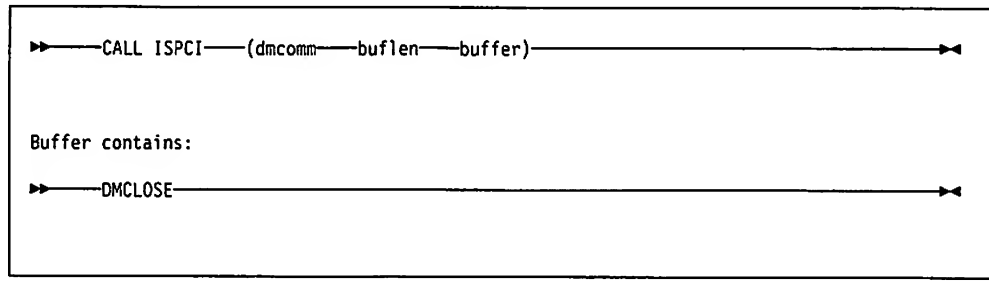
```
   VAR [PUBLIC]
      BUFFER   :STRING(512);
      BUFLEN   :INTEGER4;


      COPYSTR('DISPLAY PANEL(DMM001) CURSOR(NAME)',BUFFER);
      BUFLEN := 34;
      ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

In Procedures Language:

```
ADDRESS ISPCIR 'DISPLAY PANEL(DMM001) CURSOR(NAME) DMCOMM(dmc)'
```

DMCLOSE

Use the DMCLOSE service to end a dialog initiated by a DMOPEN call.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

DMCLOSE
The service name.

Description

The DMCLOSE service ends a dialog initiated by a DMOPEN call. A DMCLOSE is needed for each DMOPEN. The DMCLOSE call releases the current dialog variable pool. If it is the last DMCLOSE within a dialog, all Dialog Manager resources associated with that dialog are also released. After the DMCLOSE, the program cannot issue service requests to the Dialog Manager using the same DM communication area without another DMOPEN call.

Example

From a DM application, use the DMCLOSE service to end a given DMOPEN service call.

In C:

```
char    buffer[512];

strcpy(buffer, "DMCLOSE");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN          PIC S9(7) COMP VALUE 512.
 01 BUFFER          PIC X(512) VALUE SPACE.

    MOVE "DMCLOSE" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
      CHARACTER BUFFER*512

      INTEGER*4 BUFLEN

       BUFLEN = 7
       BUFFER = 'DMCLOSE'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm        dm_comm    <>
BFDMCLOSE     db         "DMCLOSE"
BFLDMCLOSE    dd         7

              lea        ax,dmcomm
              push       ds
              push       ax
              lea        ax,BFLDMCLOSE
              push       ds
              push       ax
              lea        ax,BFDMCLOSE
              push       ds
              push       ax
              CALL       ISPCI
```

In Pascal:

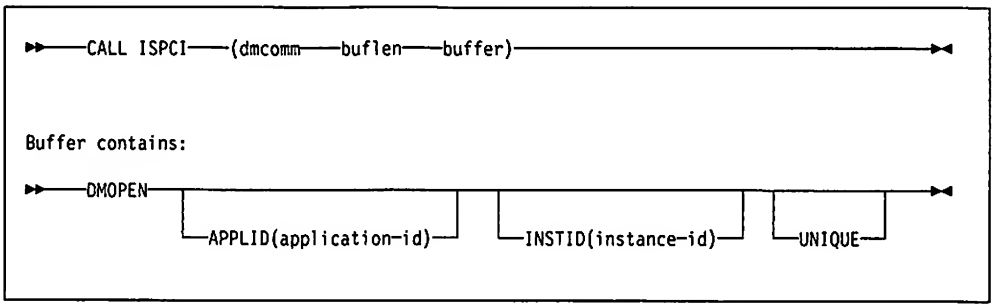
```
   VAR [PUBLIC]
      BUFFER   :STRING(512);
      BUFLEN   :INTEGER4;


      COPYSTR('DMCLOSE',BUFFER);
      BUFLEN := 7;
      ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

In Procedures Language:

```
ADDRESS ISPCIR 'DMCLOSE DMCOMM(dmc)'
```

DMOPEN

Use the DMOPEN service to activate the Dialog Manager so that the DM application can issue dialog services.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length (single-byte characters) of the buffer.

buffer
The name of a program variable that contains:

DMOPEN
The service name.

APPLID(application-id)
A 1- to 4-byte code identifying the resources to be used with this segment of the application.

- The first (or only) character must be A–Z or a–z.
- Remaining characters, if any, must be A–Z, a–z, or 0–9.
- Lowercase characters are translated to their uppercase equivalents.

The *application-id* is used to identify the application-level library definition file which, in turn, identifies the dialog element libraries to be used by the application. The *application-id* also identifies the application command table. See Chapter 8, "Using Library Services" on page 8-1 for further information.

The Dialog Manager reserves the *application-id* ISP and any *application-id* beginning with ISP.

APPLID *must* be used in the first DMOPEN for the dialog.

APPLID *may* be used in any DMOPEN service request when different dialog element libraries are needed.

INSTID(instance-id)
The 8-character *instance-id* identifying a unique occurrence or instance of a dialog.

The *instance-id* is limited to the character set 0–9, A–Z, and is generated by the Dialog Manager.

This parameter must not be specified on the first DMOPEN service call for the dialog. On the first DMOPEN call, the Dialog Manager generates and

assigns the *instance-id* and returns it to the application using the DM communication area.

On subsequent DMOPEN service calls within the dialog, the *instance-id* must be passed as a parameter on the call.

UNIQUE

The UNIQUE parameter ensures that no other active dialog can use the same APPLID. A DMOPEN specifying this parameter will be successful only if there is no other active dialog in the system using the same APPLID. Once a DMOPEN specifying this parameter is successful, any other DMOPEN in the system that attempts to use the same APPLID will fail.

For example, the following pseudocode illustrates using the UNIQUE parameter on a DMOPEN call in a way that is not valid.

```
PROG 1

  DMOPEN APPLID(AAAA) UNIQUE
  DISPLAY PANEL(A)
  DosCreateThread
    /* to start PROG 2 */
        .
        .
        .

PROG 2 on Thread 2

  DMOPEN APPLID(AAAA)
```

Since a dialog using the *application-id* AAAA exists, and UNIQUE was specified, the second DMOPEN must use a different *application-id*.

Description

The first dialog service call that a DM application issues must be the DMOPEN service. Every DM application must perform a DMOPEN. The first DMOPEN call for the DM application establishes a dialog. See "Beginning and Ending the Dialog Manager Application" on page 2-1 for a description of a dialog.

Every DMOPEN service call must specify a unique DM communication area.

If the INSTID parameter is not specified on the DMOPEN service, a new dialog instance is created and the Dialog Manager creates and assigns a unique *instance-id*. The *instance-id* identifies a dialog and is placed in the DM communication area. A new dialog can only be created by the first DMOPEN in an OS/2 thread. Therefore, the first DMOPEN service call should not specify the INSTID parameter.

Specifying an *instance-id* on the DMOPEN service causes the Dialog Manager to associate this DMOPEN service with an active dialog. Except for the first DMOPEN service call in a thread, all subsequent nested DMOPEN service calls must specify the INSTID parameter using the *instance-id* returned in the DM communication area on the first DMOPEN. The new DM communication area will receive the same *instance-id* as specified with the INSTID parameter. An advantage to using nested DMOPEN service calls is to be able to change the *application-id* or to obtain a new variable pool.

Note: A new variable pool is always established as a result of the DMOPEN service unless the application is a Procedures Language program.

See "Multiple DMOPEN Service Calls" on page 13-1 for more information.

Every DMOPEN requires a DMCLOSE to free the resources used by the Dialog Manager. DMOPEN, and its corresponding DMCLOSE service call, must be properly nested. Once a DMOPEN service is performed, the DM communication area specified should be used on all DM services until its corresponding DMCLOSE. Thus, there is only one DM communication area active within the dialog at any given time. The active DM communication area is the one associated with the most recent DMOPEN service.

APPLID is an optional parameter. It is required on the first DMOPEN service call for a dialog. Subsequent DMOPEN service calls within a dialog do not have to specify the APPLID parameter. If APPLID is not specified, the currently defined libraries containing dialog elements will continue to be used. If APPLID is specified, the set of DM libraries containing dialog elements that is identified in the application library definition file (xxxxLIB.APP) will be used.

Note: No variable substitution is allowed.

Example

The DMOPEN service activates the Dialog Manager.

In C:

```
char    buffer[512];

strcpy(buffer, "DMOPEN APPLID(SAMP)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN           PIC S9(7) COMP VALUE 512.
 01 BUFFER           PIC X(512) VALUE SPACE.

    MOVE "DMOPEN APPLID(SAMP)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
CHARACTER BUFFER*512

INTEGER*4 BUFLEN

 BUFLEN = 19
 BUFFER = 'DMOPEN APPLID(SAMP)'
 CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm      dm_comm   <>
BFDMOPEN    db        "DMOPEN APPLID(SAMP)"
BFLDMOPEN   dd        19

            lea       ax,dmcomm
            push      ds
            push      ax
            lea       ax,BFLDMOPEN
            push      ds
            push      ax
            lea       ax,BFDMOPEN
            push      ds
            push      ax
            CALL      ISPCI
```

In Pascal:

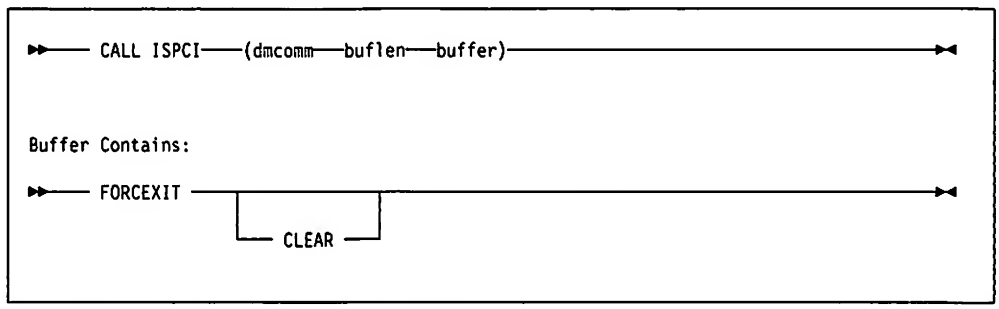
```
VAR [PUBLIC]
   BUFFER   :STRING(512);
   BUFLEN   :INTEGER4;


   COPYSTR('DMOPEN APPLID(SAMP)',BUFFER);
   BUFLEN := 19;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

In Procedures Language:

```
ADDRESS ISPCIR 'DMOPEN APPLID(SAMP) DMCOMM(dmc)'
```

FORCEXIT

Use the FORCEXIT service from a non-Dialog Manager thread to cause a Dialog Manager thread to exit from its DISPLAY service. Use the FORCEXIT CLEAR service from a Dialog Manager thread to clear any pending FORCEXIT request.



dmcomm
The name of a DM application variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a DM application variable that contains:

FORCEXIT
The service name.

CLEAR
Specifies that the Dialog Manager clears any pending FORCEXIT request.

If no option is specified, the Dialog Manager exits the current display.

Description

The FORCEXIT service is used by a non-Dialog Manager thread to force a Dialog Manager thread to exit from its DISPLAY service. When using this form of the FORCEXIT service, Dialog Manager assumes that the DM communication area parameter was created by a DMOPEN on another thread.

The FORCEXIT CLEAR service is used by a Dialog Manager thread to remove any pending FORCEXIT request made by a non-Dialog Manager thread after the user cancelled the action by selecting a choice on the displayed panel.

Note: No variable substitution is allowed.

See "Forced Display Exit" on page 12-1 for more details.

Example

The FORCEXIT CLEAR service is used by a Dialog Manager thread to remove any pending FORCEXIT request made by a non-Dialog Manager thread after the user cancelled the action by selecting a choice on the displayed panel.

In C:

```
char    buffer[512];

strcpy(buffer, "FORCEXIT CLEAR");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN          PIC S9(7) COMP VALUE 512.
 01 BUFFER          PIC X(512) VALUE SPACE.

    MOVE "FORCEXIT CLEAR" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
CHARACTER BUFFER*512

INTEGER*4 BUFLEN

 BUFLEN = 14
 BUFFER = 'FORCEXIT CLEAR'
 CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm        dm_comm   <>
BFFORCEXIT    db        "FORCEXIT CLEAR"
BFLFORCEXIT   dd        14

              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLFORCEXIT
              push      ds
              push      ax
              lea       ax,BFFORCEXIT
              push      ds
              push      ax
              CALL      ISPCI
```

In Pascal:

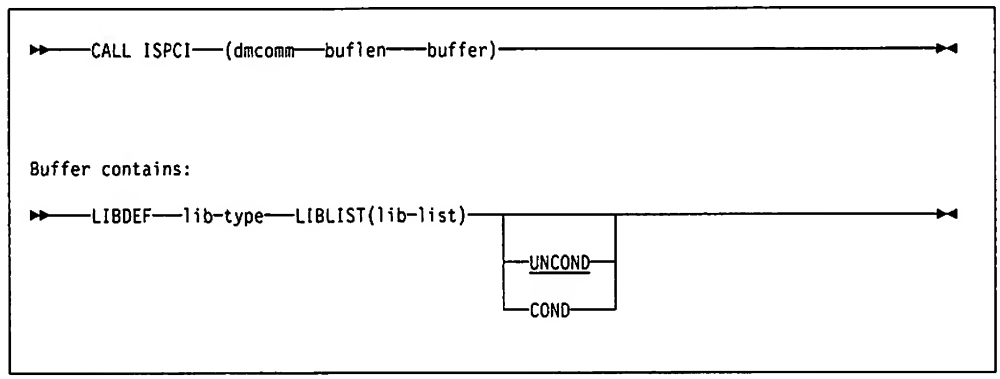
```
VAR [PUBLIC]
   BUFFER   :STRING(512);
   BUFLEN   :INTEGER4;


   COPYSTR('FORCEXIT CLEAR',BUFFER);
   BUFLEN := 14;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

In Procedures Language:

```
ADDRESS ISPCIR 'FORCEXIT CLEAR DMCOMM(dmc)'
```

LIBDEF

Use the LIBDEF service to define a set of libraries dynamically.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

LIBDEF
The service name.

lib-type
Indicates which type of library definition is being made. The following types of libraries can be specified:

LIBRARY The files containing panels, messages, key mapping lists, and icons can be included in this type.

The LIBRARY file is searched for in the current directory and the paths specified in the OS/2 DPATH environment variable.

HELP The files containing help panels are included in this type.

The HELP file is searched for in the current directory and the paths specified in the OS/2 HELP environment variable.

LIBLIST(lib-list)
A list of library file names. This parameter can be a single name or a list of names enclosed in parentheses and separated by commas, or blanks, or both.

Note: Optionally, you can use the less-than (<) and greater-than (>) symbols instead of parentheses. If the library file name contains one or more parenthesis, you *must* use the less-than and greater-than symbols in the LIBLIST specification.

If the library file name contains one or more blanks or commas, the file name must be enclosed in quotation marks (""), like this:

```
"FILE NAME.DTL"
```

The number of names in the *lib-list* is limited only by the 512-byte buffer.

You must specify the complete file name, but not the path, of the library file.

The LIBDEF service does not check for the existence of the library file or for the validity of the library file name.

UNCOND
The library definition should be accepted regardless of the existence of an active library definition in this dialog for the specified library type. This is the default parameter.

COND
The library definition should be accepted only if there is no active library definition in this dialog for the specified type.

Description

The LIBDEF service allows you to replace the library definition set by the LIBDEF command in the application library definition file.

The libraries defined by the LIBDEF service call remain in effect while the associated DMOPEN call is active. If multiple DMOPEN calls share the same *application-id* and *instance-id*, the library definition set by the LIBDEF service is also shared.

Each LIBDEF service request replaces an existing library definition.

See Chapter 8, "Using Library Services" on page 8-1 for a description of using the LIBDEF service in your DM application.

Example

In this example, the LIBDEF service call instructs the program to use the libraries *PGMLIB.DTL* and *TESTLIB.DTL*. These libraries are both of LIBRARY type and exist in the current directory or the paths specified in the OS/2 DPATH environment variable.

In C:

```
char    buffer[512];

strcpy(buffer, "LIBDEF LIBRARY LIBLIST(TESTLIB.DTL,PGMLIB.DTL)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN           PIC S9(7) COMP VALUE 512.
 01 BUFFER           PIC X(512) VALUE SPACE.

    MOVE "LIBDEF LIBRARY LIBLIST(TESTLIB.DTL,PGMLIB.DTL)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
CHARACTER BUFFER*512

INTEGER*4 BUFLEN

 BUFLEN = 46
 BUFFER = 'LIBDEF LIBRARY LIBLIST(TESTLIB.DTL,PGMLIB.DTL)'
 CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm      dm_comm   <>
BFLIBDEF    db        "LIBDEF LIBRARY LIBLIST(TESTLIB.DTL,PGMLIB.DTL)"
BFLLIBDEF   dd        46

            lea       ax,dmcomm
            push      ds
            push      ax
            lea       ax,BFLLIBDEF
            push      ds
            push      ax
            lea       ax,BFLIBDEF
            push      ds
            push      ax
            CALL      ISPCI
```

In Pascal:

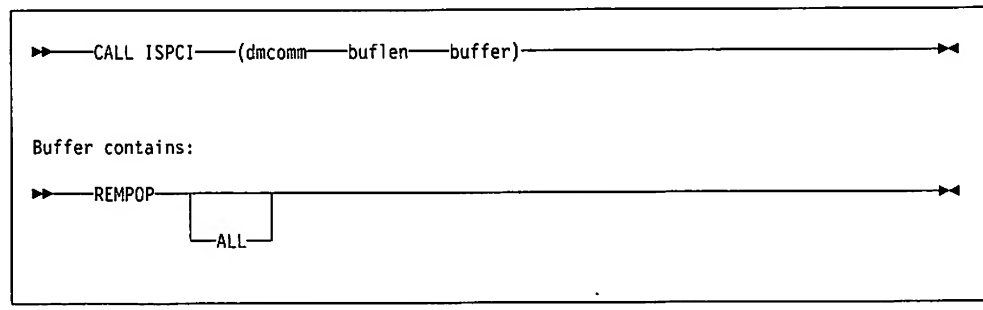
```
VAR [PUBLIC]
   BUFFER   :STRING(512);
   BUFLEN   :INTEGER4;


   COPYSTR('LIBDEF LIBRARY LIBLIST(TESTLIB.DTL,PGMLIB.DTL)',BUFFER);
   BUFLEN := 46;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

In Procedures Language:

```
ADDRESS ISPCIR 'LIBDEF LIBRARY LIBLIST(TESTLIB.DTL,PGMLIB.DTL)
       DMCOMM(dmc)'
```

REMPOP

Use the REMPOP service to remove pop-up windows created by ADDPOP service calls.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

REMPOP
The service name.

ALL
Indicates that the Dialog Manager is to remove all pop-up windows that were created in the current dialog. If you do not specify ALL, only one level of pop-up window is removed.

Description

The REMPOP service removes pop-up windows created by ADDPOP service calls. After calling the REMPOP service, any subsequent DISPLAY service call will either display a panel in the DM application primary window or in a higher level pop-up window, if one is active.

If the optional ALL parameter is included in the service call, all pop-up windows within the current dialog are removed. Without the ALL parameter, only one level of pop-up window is removed.

For a complete description of using the REMPOP service to remove pop-up windows, see "Using Pop-up Windows" on page 2-3.

Example

The REMPOP service call removes the pop-up window that would be displayed in the following example.

In C:

```
char    buffer[512];

strcpy(buffer, "DISPLAY PANEL(DMC301)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);

strcpy(buffer, "ADDPOP");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);

strcpy(buffer, "DISPLAY PANEL(DME301)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);

strcpy(buffer, "REMPOP");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);

strcpy(buffer, "DISPLAY PANEL(DME301)");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN           PIC S9(7) COMP VALUE 512.
 01 BUFFER           PIC X(512) VALUE SPACE.

    MOVE "DISPLAY PANEL(DMC301)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

    MOVE "ADDPOP" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

    MOVE "DISPLAY PANEL(DME301)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

    MOVE "REMPOP" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.

    MOVE "DISPLAY PANEL(DME301)" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
      CHARACTER BUFFER*512

      INTEGER*4 BUFLEN

       BUFLEN = 21
       BUFFER = 'DISPLAY PANEL(DMC301)'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)

       BUFLEN = 6
       BUFFER = 'ADDPOP'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)

       BUFLEN = 21
       BUFFER = 'DISPLAY PANEL(DME301)'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)

       BUFLEN = 6
       BUFFER = 'REMPOP'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm      dm_comm   <>
BFDMC301    db        "DISPLAY PANEL(DMC301)"
BFLDMC301   dd        21
BFADP       db        "ADDPOP"
BFLADP      dd        6
BFDME301    db        "DISPLAY PANEL(DME301)"
BFLDME301   dd        21
BFRMP       db        "REMPOP"
BFLRMP      dd        6
BFDME301    db        "DISPLAY PANEL(DME301)"
BFLDME301   dd        21

            lea       ax,dmcomm
            push      ds
            push      ax
            lea       ax,BFLDMC301
            push      ds
            push      ax
            lea       ax,BFDMC301
            push      ds
            push      ax
            CALL      ISPCI

            lea       ax,dmcomm
            push      ds
            push      ax
            lea       ax,BFLADP
            push      ds
            push      ax
            lea       ax,BFADP
            push      ds
            push      ax
            CALL      ISPCI

            lea       ax,dmcomm
            push      ds
            push      ax
            lea       ax,BFLDME301
            push      ds
            push      ax
            lea       ax,BFDME301
            push      ds
            push      ax
            CALL      ISPCI

            lea       ax,dmcomm
            push      ds
            push      ax
            lea       ax,BFLRMP
            push      ds
            push      ax
            lea       ax,BFRMP
            push      ds
            push      ax
            CALL      ISPCI
```

```
lea     ax,dmcomm
push    ds
push    ax
lea     ax,BFLDME301
push    ds
push    ax
lea     ax,BFDME301
push    ds
push    ax
CALL    ISPCI
```

In Pascal:

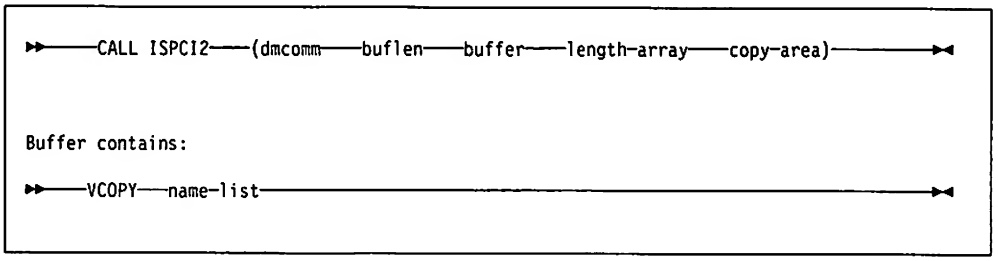
```
VAR [PUBLIC]
   BUFFER   :STRING(512);
   BUFLEN   :INTEGER4;

   COPYSTR('DISPLAY PANEL(DMC301)',BUFFER);
   BUFLEN := 21;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

   COPYSTR('ADDPOP',BUFFER);
   BUFLEN := 6;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

   COPYSTR('DISPLAY PANEL(DME301)',BUFFER);
   BUFLEN := 21;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

   COPYSTR('REMPOP',BUFFER);
   BUFLEN := 6;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);

   COPYSTR('DISPLAY PANEL(DME301)',BUFFER);
   BUFLEN := 21;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

In Procedures Language:

```
ADDRESS ISPCIR 'DISPLAY PANEL(DMC301) DMCOMM(dmc)'

ADDRESS ISPCIR 'ADDPOP DMCOMM(dmc)'

ADDRESS ISPCIR 'DISPLAY PANEL(DME301) DMCOMM(dmc)'

ADDRESS ISPCIR 'REMPOP DMCOMM(dmc)'

ADDRESS ISPCIR 'DISPLAY PANEL(DME301) DMCOMM(dmc)'
```

VCOPY

Use the VCOPY service to obtain a copy of dialog variables for your DM application.



Note: For Procedures Language programs, the *length-array* and *copy-area* parameters are omitted. See "Procedures Language Syntax Example" on page 14-3 for an example of Procedures Language syntax.

dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

VCOPY
The service name.

name-list
The names of dialog variables to be copied. Use the standard *name-list* format. Variable names can be subscripted if they were defined with the *number-of-occurrences* parameter on the VDEFINE service. You can specify the *name-list* parameter as a single variable name, or as a list enclosed in parentheses. See "Passing Dialog Variables as Parameters" on page 4-3 for more information about the *name-list* parameter.

length-array
An array of signed 4-byte binary integers. Each element of the array contains the maximum number of bytes in the *copy-area* that are to be used for storing the value of the corresponding dialog variable. The array can consist of a single item. The VCOPY service sets each element of the array to the number of bytes of data copied, not including any trailing blanks, for the corresponding variable. If a variable is not found, the corresponding length field is set to zero.

Note: For Procedures Language programs, this parameter is omitted as the Dialog Manager will use the current lengths of the dialog variables specified in the *name-list*.

copy-area
The name of an area in storage that is to receive the copied values. If more than one variable is to be copied, the *copy-area* is divided into several parts. The length of each part is defined by the values in the *length-array*.

Note: For Procedures Language programs, this parameter is omitted as the Dialog Manager will copy the dialog variables specified in the *name-list* into the Procedures Language variable pool using the specified names.

Description

The VCOPY service allows a DM application to obtain a copy of dialog variables. The copied data is in character-string format and is copied into the DM application's own storage.

VCOPY is supported in Procedures Language programs to provide access to Dialog Manager system variables. Although the Dialog Manager does not restrict the use of VCOPY in Procedures Language only to DM system variables, it is not necessary to copy other dialog variables, as they are already in the Procedures Language pool implicitly.

You must supply an array of maximum lengths on input. Its values map the *copy-area* that must be supplied to receive the data. You must first allocate storage for the data, and then call the VCOPY service. This passes the address and lengths for the storage area into which the Dialog Manager copies the data. The Dialog Manager then sets the length array with the data lengths that it finds.

For Procedures Language programs, the Dialog Manager copies the values specified in the *name-list* directly into the Procedures Language pool using the same names as in the *name-list*, and the current lengths of the dialog variables. Therefore, the *length-array* and *copy-area* parameters are not used in Procedures Language VCOPY service calls.

If the Dialog Manager does not find a variable that appears in the *name-list*, VCOPY processing continues and issues a non-zero return code.

See Chapter 7, "Using Dialog Variables, the Dialog Variable Pool, and Variable Services" on page 7-1 for more information on using the VCOPY service.

See "Calling Services in Procedures Language" on page 4-20 for more information on using the VCOPY service in Procedures Language programs.

Example

Copy the values in dialog variables *STR1* and *STR2* to a *copy-area* named *STRDATA* that contains two fields, *STRDATA1* and *STRDATA2*, of 10 bytes each and are initialized to blanks. *LNG* is an integer array of 2 elements and both elements are initialized to 10. Dialog variable *STR1* contains the characters ABC and dialog variable *STR2* contains the characters DEFG.

The status of the variables after the service call has completed is shown below. Blanks are represented by the letter *b*.

- *STRDATA1* contains 'ABCbbbbbbb'
- *STRDATA2* contains 'DEFGbbbbbb'
- *LENARRAY(1)* contains 3
- *LENARRAY(2)* contains 4.

In C:

```
char    buffer[512];
long    lenarray[2];

struct
{
  char  strdata1[10];
  char  strdata2[10];
} strdata;

lenarray[0] = sizeof(strdata.strdata1);
lenarray[1] = sizeof(strdata.strdata2);
strcpy(buffer, "VCOPY (STR1 STR2)");
ISPCI2(&dmcomm,(long)strlen(buffer),buffer,lenarray,&strdata);
```

In COBOL:

```
 01 BUFLEN           PIC S9(7) COMP VALUE 512.
 01 BUFFER           PIC X(512) VALUE SPACE.
 01 LENARRAY.
     05 VARLEN       PIC S9(7) COMP OCCURS 2 TIMES.

 01 STRDATA.
     05 STRDATA1     PIC X(10).
     05 STRDATA2     PIC X(10).

     MOVE 10 TO VARLEN(1).
     MOVE 10 TO VARLEN(2).


     MOVE "VCOPY (STR1 STR2)" TO BUFFER.
     CALL 'ISPCI2' USING DMCOMM BUFLEN BUFFER LENARRAY STRDATA.
```

In FORTRAN:

```
      CHARACTER BUFFER*512
      CHARACTER*10 STRDATA(2)
      INTEGER*4 BUFLEN,LENARRAY(2)

      LENARRAY(1) = 10
      LENARRAY(2) = 10

       BUFLEN = 17
       BUFFER = 'VCOPY (STR1 STR2)'
       CALL ISPCI2(DMCOMM,BUFLEN,BUFFER,LENARRAY,STRDATA)
```

In MASM:

```
dmcomm          dm_comm    <>
BFVCOPY         db         "VCOPY (STR1 STR2)"
BFLVCOPY        dd         17

                EVEN
type_strdata    STRUC
STRDATA1        db         10 dup(?)
STRDATA2        db         10 dup(?)
type_strdata    ENDS
strdata         type_strdata   <>

                EVEN
type_strdata_L  STRUC
STRDATA1_L      dd         10
STRDATA2_L      dd         10
type_strdata_L  ENDS
strdata_L         type_strdata_L   <>

                lea        ax,dmcomm
                push       ds
                push       ax
                lea        ax,BFLVCOPY
                push       ds
                push       ax
                lea        ax,BFVCOPY
                push       ds
                push       ax
                lea        ax,strdata_L
                push       ds
                push       ax
                lea        ax,strdata
                push       ds
                push       ax

                CALL       ISPCI2
```

In Pascal:

```
TYPE
     LNG=RECORD
        LNG1     :INTEGER4;
        LNG2     :INTEGER4;
     END;

     USER=RECORD
       STRDATA1   :STRING(10);
       STRDATA2   :STRING(10);
     END;

     U_ARRAY = ARRAY [1..5] of USER;

VAR [PUBLIC]
     BUFFER   :STRING(512);
     BUFLEN   :INTEGER4;

     STRDATA  :U_ARRAY;
     LENARRAY :LNG;

 BEGIN

     LENARRAY.LNG1 := 10;
     LENARRAY.LNG2 := 10;

     COPYSTR('VCOPY (STR1 STR2)',BUFFER);
     BUFLEN := 17;
     ISPCI2 (DMCOMM, BUFLEN, ADS BUF1, ADS LENARRAY ADS STRDATA);
```

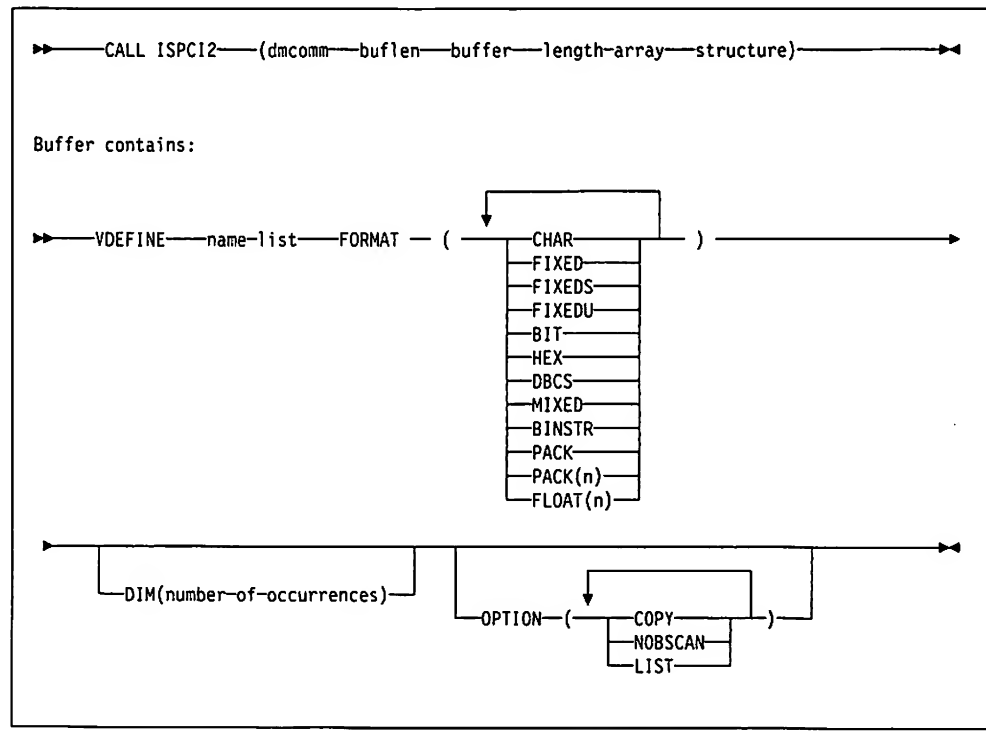
Procedures Language Example

This Procedures Language example copies the Dialog Manager system variables ZIDATE and ZITIME to the Procedures Language variable pool.

```
ADDRESS ISPCIR 'VCOPY (ZIDATE ZITIME) DMCOMM(dmc)'
```

VDEFINE

Use the VDEFINE service to explicitly define dialog variables to the Dialog Manager.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

VDEFINE
The service name.

name-list
The symbolic name or names that the Dialog Manager uses when referencing the specified variables. Use the standard *name-list* format. Subscripted variable names are not allowed.

An asterisk (*) in the *name-list*, in conjunction with an asterisk (*) in the FORMAT parameter, specifies that the storage represented by the corresponding physical length in the *length-array* parameter is to be skipped when calculating the address of the next name in the *name-list*. When this facility is used, LIST must be specified in the OPTION parameter. When you use the DIM parameter, program storage at the end of an array or multiple occurrence structure (unless defined as a dialog variable) must be specified with the "*" placeholder, so that Dialog Manager will know where the first occurrence of the array or structure ends and the next begins. You must ensure that the placeholder length is correct.

See "Passing Dialog Variables as Parameters" on page 4-3 for more information about the *name-list* parameter.

FORMAT(format)
The data conversion format.

When LIST is specified in the OPTION parameter, the FORMAT parameter is a list of format values, one for each variable in the *name-list*. Each element of this list defines the data format of the variable in the corresponding position in the *name-list*. An asterisk (*) in the format list has a special meaning as described under *name-list* above.

Valid data conversion formats for the FORMAT parameter are:

CHAR
Character string. Within the variable, the data is left-justified and padded on the right with blanks.

The Dialog Manager does not perform data conversion or check for valid characters when retrieving and storing a CHAR variable.

CHAR dialog variables can have data lengths from 1 to 32767 bytes.

FIXED
Fixed binary integer, externally represented by the characters 0 – 9.

Fixed variables can have a length of 1 – 4 bytes. Fixed variables that have a length of 4 bytes are treated as signed, externally represented by the absence or presence of a leading minus sign (-). They can also have a null value, which the Dialog Manager stores as the maximum negative number.

Fixed variables that have a length of less than 4 bytes are treated as unsigned. For these variables, a null value is stored as binary zeros and cannot be distinguished from a zero value.

The Dialog Manager converts FIXED dialog variables to character form when necessary; for example, when displaying them.

FIXEDS
Fixed signed integer, externally represented by the characters 0 – 9. FIXEDS variables can have a length of 1, 2, or 4 bytes. The sign is externally represented by the absence or presence of a leading minus sign (-). FIXEDS dialog variables can have a null value, which the Dialog Manager stores as the maximum negative number.

The Dialog Manager converts FIXEDS dialog variables to character form when necessary; for example, when displaying them.

FIXEDU
Fixed unsigned integer, externally represented by the characters 0 – 9.

FIXEDU variables can have a length of 1, 2, or 4 bytes. For these variables, a null value is stored as binary zeros, and cannot be distinguished from a zero value.

The Dialog Manager converts FIXEDU dialog variables to character form when necessary; for example, when displaying them.

BIT
A sequence of bytes, with each bit externally represented by the character 0 or 1. Within the variable, the data is left-justified and padded on the right with binary zeros. For these variables, the Dialog Manager stores a null value as binary zeros.

When a variable service retrieves a BIT variable, the service returns a string of characters (0 – 1).

BIT variables can have a length of 1 – 4095 bytes. The length of a BIT dialog variable is specified as the number of bytes, *not* the number of bits.

HEX

A sequence of bytes, with each byte externally represented as a pair of the characters 0 – 9 and A – F. Within the variable, the data is left-justified and padded on the right with binary zeros. For these variables, the Dialog Manager stores a null value as binary zeros.

When a variable service retrieves a HEX variable, the service returns a string of characters (0 – 9 and A – F).

HEX variables can have a length of 1 – 16383 bytes. The length of a HEX dialog variable is specified as the number of bytes, *not* the number of hexadecimal characters.

DBCS

Double-byte character set string. Within the variable, the data is left-justified and padded on the right with DBCS blanks. The contents of the variable must contain an even number of bytes.

The Dialog Manager does not perform data conversion or check for valid characters when retrieving and storing a DBCS variable.

DBCS variables can have a length of 2 – 32766 bytes.

MIXED

Mixed string consisting of double-byte and single-byte characters. Within the variable, the data is left-justified and padded on the right with single-byte blanks.

The Dialog Manager does not perform data conversion when retrieving and storing a MIXED variable.

MIXED variables can have a length of 1 – 32767 bytes.

BINSTR – Null Terminated String

The Dialog Manager provides the binary string data format to support DM applications written in the C language. It treats the data format exactly like the character string format, except in the manner in which it handles the padding of character strings. When a variable defined as BINSTR is updated in the dialog variable pool, the Dialog Manager pads with binary zeros, rather than blanks. This is desirable within C programs, because the C language uses binary zeros to mark the end of a character string.

In updating this type of variable, the Dialog Manager stores only up to "length – 1" amount of significant data and places a null terminator in the last position.

PACK or PACK(n) – COBOL Packed Decimal

The packed-decimal data format provides support for COBOL and corresponds to a SAA COBOL COMP-3 data type. Packed-decimal variables consist of right-justified numeric values stored so that each decimal digit takes up one-half byte. All bytes contain two decimal digits, except for the last byte in the variable. The last byte consists of the least significant decimal digit followed by a representation of the sign, always resulting in an odd number of digits.

The valid values to represent the sign are the hexadecimal digits C for positive and D for negative. If the sign is any other hexadecimal digit, the value is considered to be unsigned.

The length that you specify for this type must equal the total number of bytes of program storage that the variable uses. PACK variables can have a length of 1 – 10 bytes.

When you define a variable as having a PACK(*n*) data format, *n* specifies the number of digits to appear to the right of the assumed decimal point. For example, the value of a variable when defined is 12345. The assumed decimal position would occur before the 1 if defined as PACK(5), after the 1 if defined as PACK(4), after the 2 if defined as PACK(3), and so on.

Variables defined as PACK or PACK(*n*) are converted to character representation when retrieved from the dialog variable pool. If the variable is defined as PACK(*n*), and *n* is greater than zero, the converted number will contain a decimal separator followed by *n* digits after the assumed decimal point.

When a variable defined as PACK(*n*) is updated in the dialog variable pool, the Dialog Manager will pad the variable with zeros or truncate on either end to ensure the variable contains the correct number of digits to the right of the assumed decimal separator.

The value of *n* must be in the range 0 – 18.

FLOAT(n) – Floating Point

The floating point data format is used for variables consisting of numeric values stored in characteristic and mantissa form.

The Institute of Electrical and Electronics Engineers (IEEE) standard format is used as the internal representation.

Floating point numbers are processed as real numbers. A single precision number is processed as a 32-bit real number and can have 6 or 7 significant digits. The length of a single precision number must be specified as 4. A double precision number is processed as a 64-bit real number and can have 15 or 16 significant digits. The length of a double precision number must be specified as 8.

The format not only specifies the variable type, but it also specifies the number of digits to appear to the right of the decimal separator when converted to character representation. A format of FLOAT(1) indicates a floating point value with one digit to the right of the decimal separator. A format of FLOAT(0) indicates a floating point value with the decimal separator omitted. The value of *n* for a single precision number must be 0 – 25. The value of *n* for double precision numbers must be 0 – 50.

Floating point numbers are converted to character representation when retrieved from the dialog variable pool. When converted, the value may be rounded up if the number contains more fractional digits than are to appear to the right of the decimal separator. The value will be rounded if the next fractional digit that is not to appear to the right of the separator is 5 or greater. For example, the value of a floating point number when defined is 123.56. The converted character representation of the number would be 123.56 if defined as FLOAT(2), 123.6 if defined as FLOAT(1), and 124 if defined as FLOAT(0).

The maximum length of the converted number, including sign and decimal separator, is 70. Numbers that require more than 70 characters will result in an error.

If the real number is infinity, not a number, underflow, or overflow, the Dialog Manager generates a conversion error return code and the appropriate reason code.

DIM(number-of-occurrences)
DIM allows you to define arrays of a single dimension and to use subscripted dialog variable names in some variable services.

You specify the *number-of-occurrences* parameter value as a character string of decimal digits which defines the number of replications of the *name-list* the Dialog Manager can use. The *number-of-occurrences* value must be in the range 1 – 65535.

OPTION(option)
Contains the parameters COPY, NOBSCAN, or LIST. You specify these parameters in the *name-list* format.

COPY
Specifies that a dialog variable with the same name is to be used to initialize the storage of the variable being defined. The Dialog Manager searches for the variable in the dialog variable pool, and if the variable is not found there, it searches the Dialog Manager system variable area.

NOBSCAN
Specifies that any trailing blanks in the variables are to remain in the variables. NOBSCAN applies only to CHAR, DBCS, MIXED, and BINSTR variables.

In the following examples, *b* represents a blank:

```
                                Data received   Stored     Length when
                                    to store    value      moved to another
                                                           dialog element

VDEFINE ... CHAR 8 NOBSCAN      A           Abbbbbbb  8
VDEFINE ... CHAR 8 NOBSCAN      Abbb        Abbbbbbb  8
VDEFINE ... CHAR 8 NOBSCAN      Abbbbbbb    Abbbbbbb  8
VDEFINE ... CHAR 8              A           Abbbbbbb  1
VDEFINE ... CHAR 8              Abbb        Abbbbbbb  1
VDEFINE ... CHAR 8              Abbbbbbb    Abbbbbbb  1
VDEFINE ... CHAR 8 NOBSCAN      bbbbbbbb    bbbbbbbb  8
VDEFINE ... CHAR 8              bbbbbbbb    bbbbbbbb  0  null variable
```

LIST
Specifies that the dialog variables in the *name-list* to be defined are of varying formats and lengths. Each variable in the *name-list* can be any of the Dialog Manager-supported formats or lengths.

If the LIST option is not specified, all variables in the *name-list* are assumed to have the same format and length. If more than one format or length is specified, it is ignored.

When the LIST option is used, the DM application may define only some program variables in a structure. Do this by specifying an asterisk (*) as the variable name in the *name-list* and asterisk (*) in the

corresponding position in the FORMAT parameter for those portions of DM application storage that are to be skipped over (not considered by VDEFINE). The asterisk (*) placeholder in the *name-list* and the FORMAT allows the Dialog Manager to determine the address in DM application storage of the next true variable name in the *name-list*. This is determined by the corresponding length in the *length-array* parameter.

length-array
An array of signed 4-byte binary integer fields containing the lengths, in bytes, of the data areas for the dialog variable values. The array can consist of a single item. The array should contain one entry for each dialog variable in the *name-list* when the LIST option is specified. If the LIST option is not specified, all variables are assumed to have the same length.

The maximum length for FIXED, FIXEDS, and FIXEDU variables is 4 bytes. The maximum length for format type BIT is 4095 bytes. The maximum length for format type HEX is 16,383 bytes. The maximum length for CHAR and MIXED variables is 32,767 bytes; the maximum length for DBCS variables is 32,766 bytes. The maximum length of BINSTR variables is 32,767 bytes. The maximum length for FLOAT is 8 bytes. The maximum length for PACK is 10 bytes. Language limitations may also apply.

structure
The program variable whose storage the Dialog Manager uses. If a list of dialog variable names is passed in the *name-list* parameter, the storage associated with the variable contains a structure of dialog variables. The DIM parameter defines the number of array elements.

Note: The storage area addressed by a single VDEFINE must be contained within a single segment.

Description

The VDEFINE service associates a dialog variable name (for example, from the DATAVAR attribute of the DTAFLD tag) with program storage. To accomplish this, you must specify the format, name, and length of the dialog variables.

You can stack definitions for a particular dialog variable name by placing successive calls to VDEFINE for that dialog variable. This stacked VDEFINE facility is especially useful when called subroutines use the VDEFINE service for the same variable name, but with a different DM application storage location for it. Because the Dialog Manager cannot recognize a called subroutine, it does not establish a new dialog variable pool. The called subroutine must issue a VDELETE call for all variables it defines before returning to the calling code. The Dialog Manager uses the storage for the variable as it was last defined.

All VDEFINE calls should have corresponding VDELETE calls.

When you call the VDEFINE service, the Dialog Manager enters the dialog variable names into the current dialog variable pool. The dialog variables that you define with the VDEFINE service are considered to be explicitly defined. This is to distinguish them from implicitly defined variables. For information on explicit and implicit variables, see "Types of Variables" on page 7-1.

You can define a list of dialog variables with a single call to the VDEFINE service by specifying a list of names in the *name-list* parameter. Unless you specify LIST on the OPTION parameter, the program variables that correspond to the dialog

variables defined to the Dialog Manager in the *name-list* must have the same format and length.

When you define a list of dialog variables, the program variable name that is passed to the Dialog Manager in the *structure* parameter must be the name of the first variable in the program storage structure, the name of the array, or the name of the structure. If an array or a structure is defined, the Dialog Manager assumes that all contained fields are contiguous; that is, alignment is assumed to be on byte boundaries.

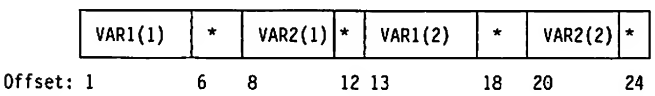
The *number-of-occurrences* parameter may be used with or without the LIST option. When an array or structure of variables is defined using this facility, any dialog variable name in the VDEFINE *name-list* may be specified with a subscript on the VCOPY and VREPLACE services. The length of the variable name with parenthesis and subscript can be more than 8 characters. The subscript must be in the range 1 – 65535 and may be specified either as a number or as a signed 4 byte binary integer program variable name that has been defined to the Dialog Manager. Examples: dialog variable ABC(5) or ABC(index). If the variable is referenced without subscript specification, the first element of the array is implied.

When using the LIST option prior to issuing the VDEFINE service request, the DM application must create an array to specify the lengths of the variables to be defined. This array defines, in sequence, the length (in bytes) of each variable. Variable names in the *name-list* that you specify on the VDEFINE request must be in the same relative positions as the corresponding length definitions in the array.

Note: VDEFINE is not supported in Procedures Language programs.

Examples

Example 1: Define two dialog variables, *VAR1* and *VAR2*, contained in a structure with a dimension of two. The name of the structure is *DIMVARS* and contains other data that will not be defined.



In C:

```
char    buffer[512];
long    lenarray[4];

struct
{
  char  var1[5];
  char  filler1[2];
  char  var2[4];
  char  filler2[1];
} dimVars[2];

lenarray[0] = sizeof(dimVars.var1);
lenarray[1] = sizeof(dimVars.filler1);
lenarray[2] = sizeof(dimVars.var2);
lenarray[3] = sizeof(dimVars.filler2);
strcpy(buffer, "VDEFINE (VAR1 * VAR2 *) FORMAT(CHAR * CHAR *) DIM(2)"
       "OPTION(LIST)");
ISPCI2(&dmcomm,(long)strlen(buffer),buffer,lenarray,dimVars);
```

In COBOL:

```
       01 BUFLEN       PIC S9(7)  COMP VALUE 512.
       01 BUFFER       PIC X(512) VALUE SPACE.
       01 LENARRAY.
          05 VARLEN    PIC S9(7)  COMP  OCCURS 4 TIMES.

       01 DIMVARS.
          05 VARDATA    OCCURS 2 TIMES.
             10 VAR1      PIC X(5).
             10 FILLER1   PIC X(2).
             10 VAR2      PIC X(4).
             10 FILLER2   PIC X(1).

           MOVE 5 TO VARLEN(1).
           MOVE 2 TO VARLEN(2).
           MOVE 4 TO VARLEN(3).
           MOVE 1 TO VARLEN(4).

           MOVE "VDEFINE (VAR1 * VAR2 *) FORMAT(CHAR * CHAR *)
       -         DIM(2) OPTION(LIST)" TO BUFFER.
           CALL 'ISPCI2' USING DMCOMM BUFLEN BUFFER LENARRAY DIMVARS.
```

In FORTRAN:

```
CHARACTER BUFFER*512,VAR1*5,VAR2*4
INTEGER*4 BUFLEN,LENARRAY(4)
CHARACTER*13 DIMVARS

EQUIVALENCE(VAR1,DIMVARS(1:5))
EQUIVALENCE(VAR2,DIMVARS(8:11))

LENARRAY(1) = 5
LENARRAY(2) = 2
LENARRAY(3) = 4
LENARRAY(4) = 1

BUFLEN = 65
BUFFER = 'VDEFINE (VAR1 * VAR2 *) FORMAT(CHAR * CHAR *) DIM(2) OPTION(LIST)'
CALL ISPCI2(DMCOMM,BUFLEN,BUFFER,LENARRAY,DIMVARS)
```

In MASM:

```
dmcomm          dm_comm    <>
BFVDEFINE       db         "VDEFINE (VAR1 * VAR2 *) FORMAT(CHAR * CHAR *) DIM(2)"
                           "OPTION(LIST)"
BFLVDEFINE      dd         65

                EVEN
type_dimVars    STRUC
VAR1            db         5 dup(?)
FILLER          db         2 dup(?)
VAR2            db         4 dup(?)
FILLER2         db         ?
type_dimVars    ENDS
dimVars         type_dimVars    <>


                EVEN
type_dimVars_L  STRUC
VAR1_L          dd         5
FILLER_L        dd         2
VAR2_L          dd         4
FILLER2_L       dd         1
type_dimVars_L  ENDS
dimVars_L       type_dimVars_L    <>


                lea        ax,dmcomm
                push       ds
                push       ax
                lea        ax,BFLVDEFINE
                push       ds
                push       ax
                lea        ax,BFVDEFINE
                push       ds
                push       ax
                lea        ax,dimVars_L
                push       ds
                push       ax
                lea        ax,dimVars
                push       ds
                push       ax

                CALL       ISPCI2
```

In Pascal:

```
TYPE
     LNG=RECORD
        LNG1    :INTEGER4;
        LNG2    :INTEGER4;
        LNG3    :INTEGER4;
        LNG4    :INTEGER4;
     END;

     STRUCT=RECORD
       VAR1       :STRING(5);
       FILL1      :STRING(2);
       VAR2       :STRING(4);
       FILL2      :STRING(1);
     END;


VAR [PUBLIC]
     BUFFER   :STRING(512);
     BUFLEN   :INTEGER4;

     DIMVARS  :STRUCT;
     LENARRAY :LNG;

 BEGIN

     LENARRAY.LNG1 := 5;
     LENARRAY.LNG2 := 2;
     LENARRAY.LNG3 := 4;
     LENARRAY.LNG4 := 1;

  COPYSTR('VDEFINE (VAR1 * VAR2 *) FORMAT(CHAR * CHAR *) DIM(2) '
          'OPTION(LIST)',BUFFER);
  BUFLEN := 65;
  ISPCI2 (DMCOMM, BUFLEN, ADS BUFFER, ADS LENARRAY, ADS DIMVARS);
```

Example 2: Define the dialog variable *MSGNAME*. The underlying program variable is *ERRMSG*.

In C:

```
char    buffer[512];
long    lenarray[4];

char  errmsg[8];

lenarray[0] = sizeof(errmsg);
strcpy(buffer, "VDEFINE (MSGNAME) FORMAT(CHAR)");
ISPCI2(&dmcomm,(long)strlen(buffer),buffer,lenarray,errmsg);
```

In COBOL:

```
01 BUFLEN       PIC S9(7)  COMP VALUE 512.
01 BUFFER       PIC X(512) VALUE SPACE.
01 LENARRAY.
   05 VARLEN    PIC S9(7)  COMP  OCCURS 4 TIMES.

01 ERRMSG       PIC X(8).


    MOVE 8 TO VARLEN(1).

    MOVE "VDEFINE (MSGNAME) FORMAT(CHAR)" TO BUFFER.
    CALL 'ISPCI2' USING DMCOMM BUFLEN BUFFER LENARRAY ERRMSG.
```

In FORTRAN:

```
CHARACTER BUFFER*512,ERRMSG*8
INTEGER*4 BUFLEN,LENARRAY

LENARRAY = 8

BUFLEN = 30
BUFFER='VDEFINE (MSGNAME) FORMAT(CHAR)'
CALL ISPCI2(DMCOMM,BUFLEN,BUFFER,LENARRAY,ERRMSG)
```

In MASM:

```
dmcomm        dm_comm   <>
BFVDEFINE2    db        "VDEFINE MSGNAME FORMAT(CHAR)"
BFLVDEFINE2   dd        28
errmsg        db        8 dup(' ')
lng           dd        8


              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLVDEFINE2
              push      ds
              push      ax
              lea       ax,BFVDEFINE2
              push      ds
              push      ax
              lea       ax,lng
              push      ds
              push      ax
              lea       ax,errmsg
              push      ds
              push      ax

              CALL      ISPCI2
```

In Pascal:

```
 VAR [PUBLIC]
      BUFFER   :STRING(512);
      BUFLEN   :INTEGER4;

      ERRMSG   :STRING(8);
      LENARRAY :INTEGER4;

  BEGIN

      LENARRAY := 8;

   COPYSTR('VDEFINE (MSGNAME) FORMAT(CHAR)',BUFFER);
   BUFLEN := 30;
   ISPCI2 (DMCOMM, BUFLEN, ADS BUFFER, ADS LENARRAY, ADS ERRMSG);
```

Example 3: Define three dialog variables, *FVAR1*, *CVAR*, and *FVAR2*, with different data formats and lengths. These variables are contained in a structure called *VARS*.

In C:

```
char    buffer[512];
long    lenarray[4];

struct
{
  long  fvar1;
  char  cvar[5];
  long  fvar2;
} vars;

lenarray[0] = sizeof(vars.fvar1);
lenarray[1] = sizeof(vars.cvar);
lenarray[2] = sizeof(vars.fvar2);

strcpy(buffer, "VDEFINE (FVAR1 CVAR FVAR2) FORMAT(FIXED CHAR FIXED)
       OPTION(LIST)");
ISPCI2(&dmcomm,(long)strlen(buffer),buffer,lenarray,&vars);
```

In COBOL:

```
01 BUFLEN       PIC S9(7)  COMP VALUE 512.
01 BUFFER       PIC X(512) VALUE SPACE.
01 LENARRAY.
   05 VARLEN    PIC S9(7)  COMP  OCCURS 4 TIMES.

01 VARS.
   05 FVAR1     PIC S9(7) COMP.
   05 CVAR      PIC X(5).
   05 FVAR2     PIC S9(7) COMP.

    MOVE 4 TO VARLEN(1).
    MOVE 5 TO VARLEN(2).
    MOVE 4 TO VARLEN(3).

    MOVE "VDEFINE (FVAR1 CVAR FVAR2) FORMAT(FIXED CHAR FIXED)
-         OPTION(LIST)" TO BUFFER.
    CALL 'ISPCI2' USING DMCOMM BUFLEN BUFFER LENARRAY VARS.
```

In FORTRAN:

```
CHARACTER BUFFER*512,CVAR*5,VARS*13
INTEGER*4 BUFLEN,LENARRAY(3),FVAR1,FVAR2

EQUIVALENCE(FVAR1,VARS(1:4))
EQUIVALENCE(CVAR,VARS(5:9))
EQUIVALENCE(FVAR2,VARS(10:13))

LENARRAY(1) = 4
LENARRAY(2) = 5
LENARRAY(3) = 4


BUFLEN = 65
BUFFER = 'VDEFINE (FVAR1 CVAR FVAR2) FORMAT (FIXED CHAR FIXED) '
         'OPTION(LIST)'
CALL ISPCI2(DMCOMM,BUFLEN,BUFFER,LENARRAY,VARS)
```

In MASM:

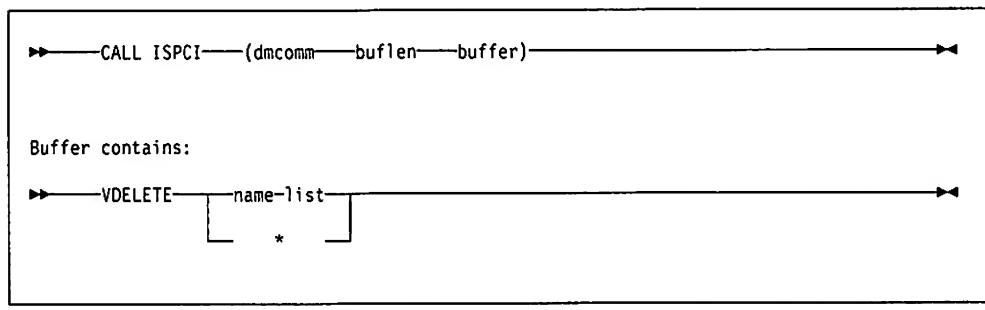
```
dmcomm          dm_comm    <>
BFVDEFINE3      db         "VDEFINE (FVAR1 CVAR FVAR2) FORMAT(FIXED CHAR FIXED) "
                db         "OPTION(LIST)"
BFLVDEFINE3     dd         64

                EVEN
type_vars       STRUC
FVAR1           dd         ?
CVAR            db         5 dup(?)
FVAR2           dd         ?
type_vars       ENDS
vars         type_vars     <>


                EVEN
type_vars_L     STRUC
FVAR1_L         dd         4
CVAR_L          dd         5
FVAR2_L         dd         4
type_vars_L     ENDS
vars_L       type_vars_L    <>

                lea        ax,dmcomm
                push       ds
                push       ax
                lea        ax,BFLVDEFINE3
                push       ds
                push       ax
                lea        ax,BFVDEFINE3
                push       ds
                push       ax
                lea        ax,vars_L
                push       ds
                push       ax
                lea        ax,vars
                push       ds
                push       ax

                CALL       ISPCI2
```

In Pascal:

```
TYPE
     LNG=RECORD
        LNG1    :INTEGER4;
        LNG2    :INTEGER4;
        LNG3    :INTEGER4;
     END;

     VARS=RECORD
       FVAR1     :STRING(5);
       CVAR      :STRING(2);
       FVAR2     :STRING(4);
     END;


VAR [PUBLIC]
     BUFFER   :STRING(512);
     BUFLEN   :INTEGER4;

     DIMVARS  :VARS;
     LENARRAY :LNG;

 BEGIN

     LENARRAY.LNG1 := 4;
     LENARRAY.LNG2 := 5;
     LENARRAY.LNG3 := 4;

  COPYSTR('VDEFINE (FVAR1 CVAR FVAR2) FORMAT(FIXED CHAR FIXED) '
          'OPTION(LIST)',BUFFER);
  BUFLEN := 64;
  ISPCI2 (DMCOMM, BUFLEN, ADS BUFFER, ADS LENARRAY, ADS VARS);
```

VDELETE

Use the VDELETE service to remove explicitly defined dialog variables from the dialog variable pool.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

VDELETE
The service name.

name-list | *
The names of the dialog variables that are to be removed from the dialog variable pool, or an asterisk. Use the standard *name-list* format. Subscripted values are not allowed.

An asterisk (*) specifies removal from the dialog variable pool of all dialog variables that the DM application previously defined. The asterisk causes the Dialog Manager to delete all stacked definitions, but does not cause deletion of implicitly defined dialog variables.

See "Passing Dialog Variables as Parameters" on page 4-3 for more information about the *name-list* parameter.

Description

The VDELETE service allows a DM application to remove previously defined dialog variables from the dialog variable pool.

See Chapter 7, "Using Dialog Variables, the Dialog Variable Pool, and Variable Services" on page 7-1 for more information on using the VDELETE service in your DM application.

Note: VDELETE is not supported in Procedures Language programs.

Example

In this example, the VDELETE service is used to remove all explicitly defined dialog variables from the dialog variable pool.

In C:

```
char    buffer[512];

strcpy(buffer, "VDELETE *");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN          PIC S9(7) COMP VALUE 512.
 01 BUFFER          PIC X(512) VALUE SPACE.

    MOVE "VDELETE *" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
CHARACTER BUFFER*512

INTEGER*4 BUFLEN

 BUFLEN = 9
 BUFFER = 'VDELETE *'
 CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

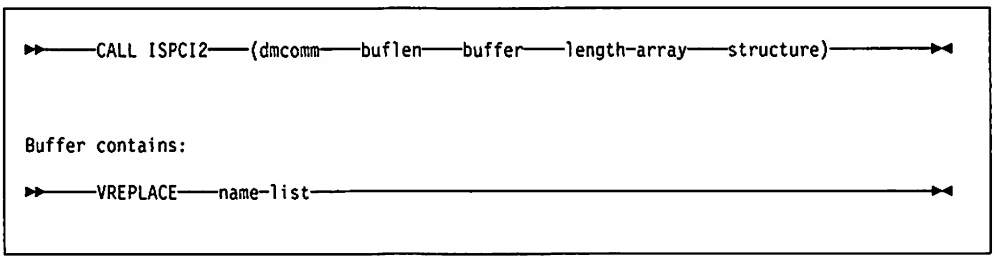
```
dmcomm        dm_comm   <>
BFVDELETE     db        "VDELETE *"
BFLVDELETE    dd        9

              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLVDELETE
              push      ds
              push      ax
              lea       ax,BFVDELETE
              push      ds
              push      ax
              CALL      ISPCI
```

In Pascal:

```
VAR [PUBLIC]
   BUFFER   :STRING(512);
   BUFLEN   :INTEGER4;


   COPYSTR('VDELETE *',BUFFER);
   BUFLEN := 9;
   ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

VREPLACE

Use the VREPLACE service to update the value of a dialog variable in the dialog variable pool.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

VREPLACE
The service name.

name-list
The names of the dialog variables whose values are to be updated. Use the standard *name-list* format. Dialog variable pool variable names can be subscripted if they were defined with the *number-of occurrences* parameter on the VDEFINE service.

See "Passing Dialog Variables as Parameters" on page 4-3 for more information about the *name-list* parameter.

length-array
An array of signed 4-byte binary integers. Each element of the array contains the number of bytes of data to be used in updating the corresponding dialog variable in the *name-list*. The array can consist of a single item.

structure
The name of a structure that contains the data with which the DM application updates the pool variables. The *length-array* maps the *structure*.

Description

The VREPLACE service allows a DM application to update the value of a variable in the dialog variable pool.

The data supplied by the application to the VREPLACE service must be character data.

The variable to be updated can be the DM application's own defined variable, if it exists, or an implicit variable associated with the DM application. If the named variable does not exist, it is created as an implicit variable.

You can specify the *name-list* parameter as a single variable name or as a list enclosed in parentheses. You must supply an array of lengths to map the area that contains the data for each of the variables.

Note: VREPLACE is not supported in Procedures Language programs.

Example

The content of the program variable *REPLDATA* is to be used to update the value of the dialog variable named *QROWS*.

In C:

```
char    buffer[512];
long    lenarray[4];
char    repldata[5];

lenarray[0] = sizeof(repldata);
memcpy(repldata, "12345", 5);

strcpy(buffer, "VREPLACE QROWS");
ISPCI2(&dmcomm,(long)strlen(buffer),buffer,lenarray,repldata);
```

In COBOL:

```
 01 BUFLEN           PIC S9(7) COMP VALUE 512.
 01 BUFFER           PIC X(512) VALUE SPACE.
 01 REPLDATA         PIC X(5) VALUE SPACE.
 01 LENARRAY.
     05 VARLEN       PIC S9(7) COMP OCCURS 4 TIMES.

     MOVE 5 TO VARLEN(1).
     MOVE "12345" TO REPLDATA.

     MOVE "VREPLACE QROWS" TO BUFFER.
     CALL 'ISPCI2' USING DMCOMM BUFLEN BUFFER LENARRAY REPLDATA.
```

In FORTRAN:

```
CHARACTER BUFFER*512,REPLDATA*5

INTEGER*4 BUFLEN,LENARRAY
REPLDATA='12345'
LENARRAY=5

 BUFLEN = 15
 BUFFER = 'VREPLACE QROWS'
 CALL ISPCI2(DMCOMM,BUFLEN,BUFFER,LENARRAY,REPLDATA)
```

In MASM:

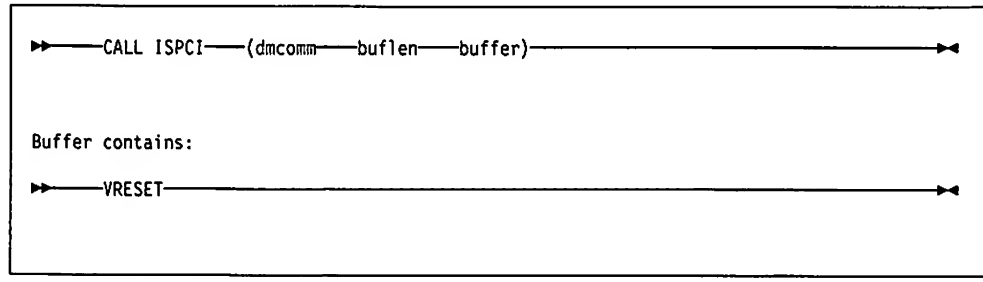
```
dmcomm        dm_comm   <>
BFVREPLACE    db        "VREPLACE QROWS"
BFLVREPLACE   dd        14
repldata      db        "12345"
repldata_L    dd        5

              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLVREPLACE
              push      ds
              push      ax
              lea       ax,BFVREPLACE
              push      ds
              push      ax
              lea       ax,repldata_L
              push      ds
              push      ax
              lea       ax,repldata
              push      ds
              push      ax

              CALL      ISPCI2
```

In Pascal:

```
VAR [PUBLIC]
   BUFFER   :STRING(512);
   BUFLEN   :INTEGER4;
   REPLDATA :STRING(5);
   LENARRAY :INTEGER4;


   COPYSTR('VREPLACE QROWS',BUFFER);
   COPYSTR('12345',REPLDATA);
   LENARRAY := 5;
   BUFLEN := 15;
   ISPCI2 (DMCOMM, BUFLEN, ADS BUFFER, ADS LENARRAY, ADS REPLDATA);
```

VRESET

Use the VRESET service to reset a DM application's dialog variable pool to empty.



dmcomm
The name of a program variable for the DM communication area. See "Service Call Format" on page 4-1 for a description of this parameter.

buflen
A signed 4-byte binary integer containing the character length of the buffer.

buffer
The name of a program variable that contains:

VRESET
The service name.

Description

The VRESET service allows a DM application to reset its dialog variable pool to empty. A Dialog Manager reference to all dialog variables, whether explicitly defined (single or stacked) or implicitly defined, using the same DM communication area will not return a value after the VRESET service.

Note: VRESET is not supported in Procedures Language programs.

Example

Reset a DM application's dialog variable pool to empty.

In C:

```
char    buffer[512];

strcpy(buffer, "VRESET");
ISPCI(&dmcomm,(long)strlen(buffer),buffer);
```

In COBOL:

```
 01 BUFLEN          PIC S9(7) COMP VALUE 512.
 01 BUFFER          PIC X(512) VALUE SPACE.

    MOVE "VRESET" TO BUFFER.
    CALL 'ISPCI' USING DMCOMM BUFLEN BUFFER.
```

In FORTRAN:

```
      CHARACTER BUFFER*512

      INTEGER*4 BUFLEN

       BUFLEN = 6
       BUFFER = 'VRESET'
       CALL ISPCI(DMCOMM,BUFLEN,BUFFER)
```

In MASM:

```
dmcomm        dm_comm   <>
BFVRESET      db        "VRESET"
BFLVRESET     dd        6

              lea       ax,dmcomm
              push      ds
              push      ax
              lea       ax,BFLVRESET
              push      ds
              push      ax
              lea       ax,BFVRESET
              push      ds
              push      ax
              CALL      ISPCI
```

In Pascal:

```
   VAR [PUBLIC]
      BUFFER   :STRING(512);
      BUFLEN   :INTEGER4;


      COPYSTR('VRESET',BUFFER);
      BUFLEN := 6
      ISPCI (DMCOMM, BUFLEN, ADS BUFFER);
```

Chapter 15. Dialog Manager Commands and Keys

The following table describes the commands supported by the Dialog Manager. The commands are arranged alphabetically.

In most cases, unless the DM application uses an application command table to override the normal behavior of a command, the processing of the command is transparent to the DM application.

Command	Description
ACTIONS	Switches the cursor to the action bar. If the cursor is already in the action bar, it switches back to the field that previously had the cursor. The Dialog Manager restricts the use of this command to the F10 key only.
BACKWARD	If the panel has a scrollable area, and the cursor is on a field within that area other than a scrollable field, such as a list field or selection list, this command displays information above that currently visible. The scroll increment is the size of the visible area minus one line. If the cursor is on a scrollable field, such as a list field or selection list, the information in the scrollable field above that currently visible would be displayed. The scroll increment is the size of the visible scrollable field minus one line. If the cursor is not on a control in the scrollable area or on a scrollable field, such as a list field or selection list, this command is ignored. **Note:** This command is most useful if mapped to a function key or an action bar pull-down choice.
CANCEL	If CANCEL is requested from a Dialog Manager message pop-up, the pop-up is removed and the cursor returns to the field that last had the cursor on the underlying panel. If CANCEL is requested from a panel displayed using the DISPLAY service call, the Dialog Manager returns: • A return code of 4. • A reason code indicating that CANCEL was requested. Reason codes indicating if validation was successful or not are returned in the Error Information section of the DM communication area. The Dialog Manager updates the variable pool regardless of the outcome of validation.

Command	Description
CHGDEFS	Displays a panel in a pop-up window listing the current default values of the ZFKA, ZMSGID, and ZPANELID system variables. In addition to the variable names and current values, the panel displays descriptions of the function provided by each variable. An ENTER action from the CHGDEFS panel removes the panel and causes the variable values to be updated. A CANCEL action removes the panel without updating the variable values.
ENTER	The ENTER command initiates the processing of the current panel. If ENTER is requested from a panel displayed using the DISPLAY service call, the Dialog Manager returns: • A return code of 0 • A reason code indicating that ENTER was requested. If variable validation fails, the panel will be redisplayed with an error message. Control will not return to the application on an ENTER action until all variable validation is successful.
EXIT	Requests that the current function be ended. If the current panel was displayed as a result of the DISPLAY service, the Dialog Manager returns: • A return code of 8. • A reason code indicating that EXIT was requested. Reason codes indicating if validation was successful or not are returned in the Error Information section of the DM communication area. The Dialog Manager updates the variable pool regardless of the outcome of validation.
EXHELP	Displays help text for an entire panel as opposed to a particular field within that panel. The text displayed is dependent on the panel that has the cursor or the panel from which help was originally requested. See Chapter 9, "Using Dialog Manager Help" on page 9-1.
FKA	Toggles the display (on or off) of the function key area for the current dialog.

Command	Description
FORWARD	If the panel has a scrollable area, and the cursor is on a field within that area other than a scrollable field, such as a list field or selection list, this command displays information below that currently visible. The scroll increment is the size of the visible area minus one line. If the cursor is on a scrollable field, such as a list field or selection list, information in the scrollable field below that currently visible would be displayed. The scroll increment is the size of the visible scrollable field minus one line. If the cursor is not on a control in the scrollable area or on a scrollable field, such as a list field or selection list, this command is ignored. **Note:** This command is most useful if mapped to a function key or an action bar pull-down choice.
HELP	Displays help text for a panel field, a message, a command, or a panel depending upon the context indicated by the cursor position. See Chapter 9, "Using Dialog Manager Help" on page 9-1.
HELPHELP	Displays a panel describing Help for help. See Chapter 9, "Using Dialog Manager Help" on page 9-1 for more information.
INDEX	Displays the help index panel. See Chapter 9, "Using Dialog Manager Help" on page 9-1.
KEYS	Displays the keys help panel named in ZKEYHELP. See Chapter 9, "Using Dialog Manager Help" on page 9-1.
LEFT	If the panel has a scrollable area, and the cursor is on a field within that area other than a scrollable field, such as a list field or selection list, this command displays information to the left of that currently visible. The scroll increment is one-third of the visible area. If the cursor is on a scrollable field, such as a list field or selection list, information in the scrollable field to the left of that currently visible would be displayed. The scroll increment is one-third of the visible scrollable field. If the cursor is not on a control in the scrollable area or on a scrollable field, such as a list field or selection list, this command is ignored. **Note:** This command is most useful if mapped to a function key or an action bar pull-down choice.
PANELID	Toggles the display of panel identifiers for the current dialog. The Dialog Manager updates the system variable ZPANELID to represent the current state of the panel ID option and saves the value in the dialog variable pool.

Command	Description
RETRIEVE	Displays previously entered commands in an active command entry field. It is only valid when the command entry field has the input focus. The commands are displayed one at a time, in a reverse sequence to which they were entered (last-in, first-out). This allows the user to easily recall a command for resubmission from the command line or to edit the command before entering it. The Dialog Manager does not retain an entered command for retrieval when the user: • Enters commands using action fields • Enters commands using keys assigned to commands • Enters commands using SELFLD pull-down choice selections. The RETRIEVE command is never placed on the retrieval stack.
RIGHT	If the panel has a scrollable area, and the cursor is on a field within that area other than a scrollable field, such as a list field or selection list, this command displays information to the right of that currently visible. The scroll increment is one-third of the visible area. If the cursor is on a scrollable field, such as a list field or selection list, information in the scrollable field to the right of that currently visible would be displayed. The scroll increment is one-third of the visible scrollable field. If the cursor is not on a control in the scrollable area or on a scrollable field, such as a list field or selection list, this command is ignored. **Note:** This command is most useful if mapped to a function key or an action bar pull-down choice.

Dialog Manager Keys

The Dialog Manager automatically adds default keys to your key mapping list if you do not specify them in your source file. The keys are:

Enter

Esc=Cancel

F3=Exit

F1=Help

Note: The F1 key cannot be overridden although help can be assigned to another key.

Chapter 16. System Variables

The system variables are described with type and pool information in the following tables. Some system variables are also discussed with the Dialog Manager service to which they apply.

- The first column gives the name of the variable.
- The second column indicates the variable type. The following abbreviations are used:

  in Input variable, set by a DM application to provide information to the Dialog Manager.

  out Output variable, set by the Dialog Manager to provide information to DM applications.

  non Non-modifiable output variable. In some cases, the variable is dynamically evaluated by the Dialog Manager; for example, ZIDATE. The DM application cannot modify the variable.

  user User-modifiable variable. Although the DM application cannot modify these variables, the values can be modified by the user using DM commands.

- The third column gives the maximum length of the variable. All system variables are in character format unless explicitly stated otherwise. Values are left justified and padded with blanks on the right.

  DBCS blanks will be used if the data is found to be DBCS.

- The fourth column gives a brief description of the variable.

Date and Time

Name	Type	Len	Description
ZDATEF	non	8	Dialog national language date format using the characters DD for day, MM for month, and YY for year. ZDATEF contains the national language date delimiter. For example, MM/DD/YY or DD.MM.YY.
ZDAY	non	2	Day of the month (2 characters).
ZIDATE	non	6	Current date with a two-digit year. The format of ZIDATE is YYMMDD (year, month, day).
ZITIME	non	4	Current time of day (hours and minutes) based on a 24-hour clock. The format is HHMM.
ZJUL	non	6	The current Julian date with a two-digit year. The format is YY.DDD.
ZMONTH	non	2	Month of the year (2 characters).
ZSTDDATE	non	8	Current date with four-digit year. The format is YYYYMMDD (year, month, day).
ZSTDJUL	non	8	The current Julian date with a four-digit year. The format is YYYY.DDD.
ZSTDTIME	non	6	Current time of day (hours, minutes, and seconds) based on the 24-hour clock. The format is HHMMSS.
ZSTDYEAR	non	4	Year (4 characters)
ZTS	non	1	Dialog national language separator character for ZITIME.
ZYEAR	non	2	Year (2 characters)

General

Name	Type	Len	Description
Z	non	0	Null variable.
ZAPPLID	non	4	Application identifier specified on the APPLID parameter of the DMOPEN service call.
ZCMD	out	255	The command name or parameters to be passed to the DM application for handling. If the action associated with the command is PASSTHRU, then ZCMD contains the command name and its parameters, if any. If the action is SETVERB, then ZCMD contains only the command parameters and ZVERB contains the command name.
ZCS	non	1-5	Currency symbol for the dialog national language. ZCS is dynamically evaluated.
ZDS	non	1	Decimal separator character for the dialog national language. ZDS is dynamically evaluated.
ZENVIR	non	32	Environment description (content is environment dependent). • 1 to 8 contain the product name, version, release, and modification level • 9 to 16 contain the operating system name • 17 to 32 contain blanks and are reserved.
ZFKA	user	1	Current setting for the function key area form. The default dialog-level value for this variable is set when the first DMOPEN for a dialog is issued. The user can modify the value of ZFKA by using the Dialog Manager CHGDEFS command or the Dialog Manager FKA command. Modifications to the value will be retained for the duration of the current dialog or until changed again within the dialog. The value is a single-character digit with the following meanings: • 0 – Do not display FKA • 1 – Display short form of FKA. The default value is 0.
ZHELPTTL	in	255	Character string to be used as the window title for the DM application's help window. If not specified, the application's primary window title will be used.
ZKEYHELP	in	8	The name of the keys help panel to be displayed when the KEYS command is run. This keys help panel remains in effect until you change the ZKEYHELP variable value.
ZMSGID	user	1	Current user setting that determines whether or not the message ID is to be displayed. The Dialog Manager sets the default dialog-level value for this variable when the first DMOPEN for a dialog is issued. The user can modify the value of ZMSGID by using the Dialog Manager CHGDEFS command. Modifications to the value will be retained for the duration of the current dialog or until changed again within the dialog. The value is a single-character digit with the following meanings: • 0 – Do not display message ID • 1 – Display message ID. The default value is 0.

Name	Type	Len	Description
ZPANELID	user	1	Current setting for the panel identifier option. The default dialog-level value for this variable is set when the first DMOPEN for a dialog is issued. The user can modify the value of ZPANELID by using the Dialog Manager CHGDEFS command or the Dialog Manager PANELID command. Modifications to the value will be retained for the duration of the current dialog or until changed again within the dialog. The value is a single-character digit with the following meanings: • 0 – Do not display panel ID • 1 – Display panel ID. The default value is 0.
ZTHS	non	1	Thousands separator for the dialog national language. ZTHS is dynamically evaluated.
ZVERB	out	8	The command name to be returned to the program for handling, after a SETVERB action has occurred. The command parameters, if any, are returned in ZCMD.
ZWINICON	in	8	The name of the icon defined using the ICON tag that is to be used on a PWS when the DM application's primary window is minimized.
ZWINTTL	in	255	Character string to be used as the window title for the DM application primary window. The maximum length of the character string is 255 bytes.

Display Characteristics

Name	Type	Len	Description
ZDBCS	non	1	The value is a single-character digit with the following meanings: • 0 – The display device does not have DBCS display capabilities. • 1 – The display device does have DBCS display capabilities.

Information Returned from Display Processing

Name	Type	Len	Description
ZCURFLD	out	8	The name of the field (or list column) that contains the cursor when the user exits the panel.
ZCURINX	out	5	The subscript of the array element that contains the cursor when the user exits the panel. This system variable will contain a zero unless the variable displayed in the field specified by ZCURFLD was defined with the VDEFINE *number-of-occurrences* parameter. The value of ZCURINX is expressed in character format.
ZCURPOS	out	5	The position of the cursor within the field specified by ZCURFLD when the user exits the panel. The value of ZCURPOS is expressed in character format. This system variable will contain a zero if the field that had the cursor was not a list field.

Chapter 17. Dialog Manager Errors

Return Codes from Services

Each service returns a numeric code, called a return code, indicating the results of the operation. Return codes are grouped into three categories:

- Normal completion (code 0).

  Return codes of this type are purely informational.

- Warning condition (codes 4 and 8).

  Indicates a warning or an exception condition that is not necessarily an error, but a condition which the DM application should be informed of. The service request will have completed.

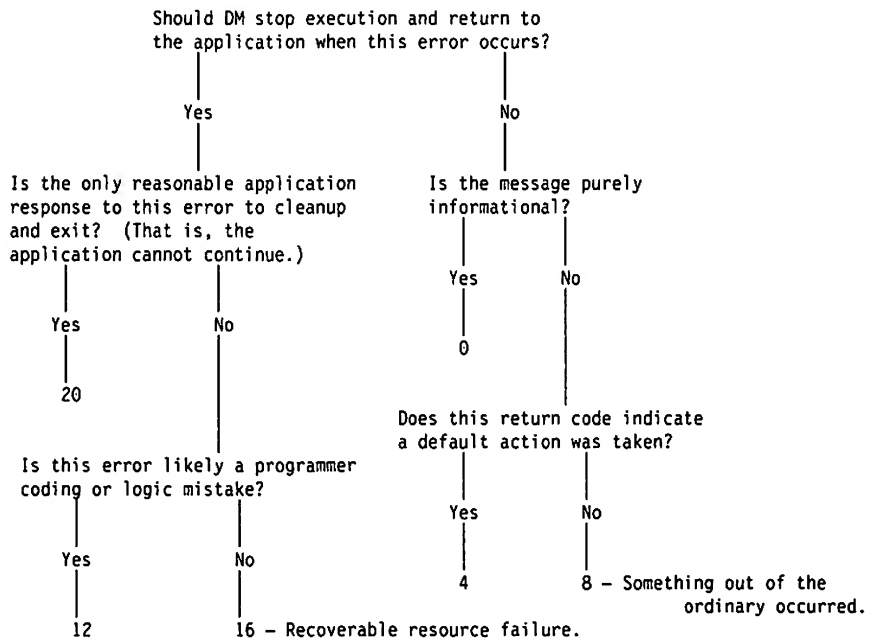
  Return code 4 indicates that a default action was taken. Return code 8 indicates that the service request completed, but something out of the ordinary occurred.

- Error condition (codes 12, 16, and 20).

  Indicates that the service found an error condition. The service may not have completed.

  Return code 12 indicates an error in program syntax or logic. Return code 16 indicates a recoverable resource failure (for example, "Out of memory" or "Unable to open file"). Return code 20 indicates an unrecoverable system failure; the application must terminate immediately.

The decision tree below shows how the DM return codes 0, 4, 8, 12, 16, and 20 are assigned.



The service passes the return code in the first 4 bytes of the DM communication area specified in the service request. The return code is a 4-byte binary integer. In addition, the return code from the DM communication area is also copied into the AX and DX registers for C, FORTRAN, MASM, and Pascal. AX contains the lower word of the return code. For COBOL, the lower word of the return code is placed in AX, while the upper word is ignored.

All return codes are passed back to the DM application for analysis and processing.

The DM application must check for and process all error conditions (return codes 0–20). For return code 20, which signifies a severe error condition, the Dialog Manager will display, if possible, an action message with the reason code, and will pass return code 20 back to the application.

Additional information about the error may be returned in bytes 17–68 of the DM communication area. This information might include the operating system error returned to the Dialog Manager when an operating system error occurs during a Dialog Manager service.

The return code from Dialog Manager to Procedures Language dialogs is returned in the Procedures Language variable *dmcomm-variable.1*, where *dmcomm-variable* is the Procedures Language stem variable specified on the service request.

Reason Codes from Services

In addition to the return code, the Dialog Manager returns a reason code. The reason code is returned in the second 4-byte field in the DM communication area.

The DM communication area contains additional reason codes that may be useful during debugging to help isolate the cause of a problem. See the section in Chapter 4, "Calling Dialog Services" on page 4-1 for the language you are using (C, COBOL, FORTRAN, MASM, or Pascal) for more details.

Errors that are returned to the Dialog Manager by Presentation Manager, and later reported to the DM application through the reason codes in the DM communication area, can be retrieved with the WinGetLastError call:

```
WinGetLastError((HAB) NULL);
```

The return value from WinGetLastError is an ERRORID and may be useful to those writing DM user controls. See "User Controls and User Exits" on page 12-8 for more information. This information is also useful to an IBM Service Representative when diagnosing a problem. Refer to the *Presentation Manager Programming Reference* for more details about WinGetLastError.

Reason codes from Dialog Manager to Procedures Language dialogs are returned in the Procedures Language variable *dmcomm-variable.2*, where *dmcomm-variable* is the Procedures Language stem variable specified on the service request.

The reason codes returned by the Dialog Manager are divided into ranges that correspond to the return code values. For each possible return code, there are 100,000 possible reason codes. Therefore, the range of possible reason codes is defined as follows:

Return Code	Reason Code range
0	0-99999
4	400000-499999
8	800000-899999
12	1200000-1299999
16	1600000-1699999
20	2000000-2099999

Note: The return code is equal to the quotient that results from dividing the reason code by 100000.

Obtaining Dialog Manager Diagnostic Information

The OS/2 Trace facility provides a means of tracing Dialog Manager activity to collect diagnostic information for an IBM Service Representative when trying to resolve a problem.

Use DM Trace in the following situations:

- If your DM application ends abnormally
- If you get an error and cannot detect its source
- If directed to do so by an IBM Service Representative.

Follow these steps to use DM Trace:

1. Put the following in your CONFIG.SYS and reboot your system:

   ```
   TRACE=OFF
   TRACEBUF=63
   SET DMTRACE=ON
   ```

2. Run your DM application to see if it ends abnormally.
3.
   a. If your DM application ends abnormally, save the OS/2 panel that appears. Press Shift and PrtSc together to print the panel. After saving the data, exit the panel by pressing the Enter key.

   b. If the DM application does not end abnormally, but has run past the point at which an error occurred, create and switch to another OS/2 session.

4. Call the following command file from an OS/2 session:

   ```
   DMLOGBUF.CMD
   ```

   DMLOGBUF.CMD will create a file named DMLOGBUF.DAT. This file contains data that can be used by an IBM Service Representative to determine the cause of the problem.

Dialog Manager Reason Codes

The following tables list all Dialog Manager reason codes. Following each reason code is a description of the error condition that caused the code to be returned to Dialog Manager.

Next, the services from which the code can be returned are listed. Some reason codes can be returned from any service. The word "Any" appears in the service column of those reason codes.

The action required, if any, to correct the error is indicated in the last column.

Return Code 0

Reason Code	Description	Service	Action Required
505	No application-level command table was found for this application ID.	DMOPEN	None
1700	The user selected Enter.	DISPLAY	Perform application-specific ENTER processing.
1701	The user issued a command that is assigned to the PASSTHRU action.	DISPLAY	Perform application-specific PASSTHRU processing.
1702	The user issued a command that is assigned to the SETVERB action.	DISPLAY	Perform application-specific SETVERB processing.
3300	The application library definition file (xxxxLIB.APP) was not in the current directory, nor in the paths specified in the DPATH environment variable. This file is optional.	DMOPEN	If the application has dependency on xxxxLIB.APP, place xxxxLIB.APP in the current directory or in the paths specified in the DPATH environment variable. Otherwise, no action is required.

Return Code 4

Reason Code	Description	Service	Action Required
400500	A key was assigned a command that is currently unavailable or unknown. No action is taken.	DISPLAY	None
400501	Received notification of a selection or keystroke that is not assigned to a choice or key. No action is taken.	DISPLAY	None
400502	Received notification of a keystroke, but no command was assigned to the key. No action is taken.	DISPLAY	None
400503	Attempted to retrieve stacked commands from an empty command stack. No action is taken.	DISPLAY	None
400504	A value being checked or translated is not valid.	DISPLAY	None
400505	The DM application's modified rows array variable contains duplicate row numbers. Duplicates are ignored.	DISPLAY	Remove duplicate MODVAR array elements.
401700	The user selected Esc=Cancel.	DISPLAY	Perform application-specific CANCEL processing.
403700	Unable to query the value of ZMSGID. The visibility of the message ID remains as it was before.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Return Code 8

Reason Code	Description	Service	Action Required
800500	Presentation Manager reported an error setting the cursor position in an entry field.	DISPLAY	Call your IBM service representative.
800501	Could not obtain ZPANELID. The panel ID is not toggled.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
800502	Could not obtain ZFKA. The FKA form is not toggled.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
800505	An error occurred during processing of the user action.	DISPLAY	A user exit is the most likely cause of the error. Determine which user exit returned this error, and correct the problem.
800506	An error occurred during processing of the user command action.	DISPLAY	A user exit is the most likely cause of the error. Determine which user exit returned this error, and correct the problem.
800507	An error occurred during processing of the user translate.	DISPLAY	A user exit is the most likely cause of the error. Determine which user exit returned this error, and correct the problem.
800508	An error occurred during processing of the user check.	DISPLAY	A user exit is the most likely cause of the error. Determine which user exit returned this error, and correct the problem.
800509	An error occurred when the area attempted to scroll.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details. If necessary, call your IBM service representative.
800523	More rows of the list field were modified than the modified rows array variable can contain.	DISPLAY	Increase the size of the MODVAR variable on the LSTFLD tag. The MODVAR variable should have at least as many elements as the smallest DATAVAR on the nested LSTCOL tags.
800524	One of the variables on the panel failed translation.	DISPLAY	None
800525	The data from the variable pool is too large for the field.	DISPLAY	Ensure that your VDEFINE service calls and VARCLASS declarations are correct.

Reason Code	Description	Service	Action Required
800526	An error was returned setting a pool variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
800527	Variable services reported an error in response to a request to get a variable from the dialog variable pool to put into a panel variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
801700	The user selected Exit.	DISPLAY	Do application-defined Exit processing.
801701	Unable to retrieve ZHELPTTL. The help window title defaults to the primary window title.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
801702	Unable to retrieve ZWINTTL.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
801703	Unable to set the FKA form for the current panel. The FKA form remains as it was before.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
801704	Unable to set the panel ID visibility for the current panel. The panel ID visibility remains as it was before.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
801705	The FORCEXIT service was called.	DISPLAY	The thread performing extended processing has completed. Continue application logic.
803700	Message text is greater than 512 character units after variable substitution. The message is truncated at the 512th character.	DISPLAY	Decrease the length of the message text in the message definition.
805300	The requested dialog variable was not found.	Any	Place the dialog variable in the variable pool using one of the following services: VDEFINE or VREPLACE.
805301	At least one dialog variable specified on the VDELETE service was not found.	VDELETE	The dialog variable to be removed from the dialog variable pool must be defined. Use the VDEFINE service to define the dialog variable in the dialog variable pool.
805302	At least one dialog variable specified on the VCOPY service was not found.	VCOPY	The dialog variable whose data is to be copied, must be placed in the dialog variable pool. Use the VDEFINE or VREPLACE service to place the dialog variable in the pool.

Reason Code	Description	Service	Action Required
805303	A dialog variable defined as FLOAT contained a non-numeric value. No data conversion was performed.	VCOPY DISPLAY	Change the value of the dialog variable so that it contains a valid real number.
805304	A dialog variable defined as FLOAT contained an indefinite value. No data conversion was performed.	VCOPY DISPLAY	Change the value of the dialog variable so that it contains a valid real number.
805305	A dialog variable defined as FLOAT contained a negative infinity value. No data conversion was performed.	VCOPY DISPLAY	Change the value of the dialog variable so that it contains a value other than a negative infinity value.
805306	A dialog variable defined as FLOAT contained a positive infinity value. No data conversion was performed.	VCOPY DISPLAY	Change the value of the dialog variable so that it contains a value other than a positive infinity value.
805307	The length of the converted floating point number exceeds the maximum size.	VCOPY DISPLAY	Reduce the magnitude of the real number contained in the dialog variable.
805308	Data truncation occurred while updating a dialog variable.	VREPLACE DISPLAY	Reduce the size of the character string used to update the dialog variable or increase the defined length of the dialog variable being updated. Ensure that your VDEFINE service calls and VARCLASS declarations are correct.
805309	Data truncation occurred while storing data into a BINSTR dialog variable.	VREPLACE DISPLAY	Reduce the size of the character string used to update the dialog variable or increase the defined length of the dialog variable being updated. Ensure that your VDEFINE service calls and VARCLASS declarations are correct.
805310	The converted FIXED number is too large to fit in the variable's defined storage.	VREPLACE DISPLAY	Reduce the magnitude of the number used to update the dialog variable.
805311	The converted FIXEDU number is too large to fit in the variable's defined storage.	VREPLACE DISPLAY	Reduce the magnitude of the number used to update the dialog variable or increase the defined length of the dialog variable being updated.
805312	The converted FIXEDS number is too large to fit in the variable's defined storage.	VREPLACE DISPLAY	Reduce the magnitude of the number used to update the dialog variable or increase the defined length of the dialog variable being updated.
805313	The converted HEX number is too large to fit in the variable's defined storage.	VREPLACE DISPLAY	Reduce the size of the data used to update the dialog variable or increase the defined length of the dialog variable being updated.

Reason Code	Description	Service	Action Required
805314	The converted PACK number is too large to fit in the variable's defined storage.	VREPLACE DISPLAY	Reduce the magnitude of the number used to update the dialog variable or increase the defined length of the dialog variable being updated.
805315	The converted BIT string is too large to fit in the variable's defined storage.	VREPLACE DISPLAY	Reduce the size of the data used to update the dialog variable or increase the defined length of the dialog variable being updated.
805316	A floating point number contained an underflow value. No data conversion was performed.	VCOPY VREPLACE DISPLAY	If the dialog variable is being updated in the dialog variable pool, increase the magnitude of the number used to update the dialog variable. If the dialog variable is being accessed from the dialog variable pool, ensure that the dialog variable does not contain an underflow value.
805317	A floating point number contained an overflow value. No conversion was performed.	VCOPY VREPLACE DISPLAY	If the dialog variable is being updated in the dialog variable pool, reduce the magnitude of the number used to update the dialog variable. If the dialog variable is being accessed from the dialog variable pool, ensure that the dialog variable does not contain an overflow value.
805318	An error occurred while converting a character string to the internal representation of a FIXED number.	VREPLACE DISPLAY	Make sure the data used to update the dialog variable contains valid characters. The valid characters used in updating a dialog variable defined as FIXED are +, -, 0−9, and the thousands separator.
805319	An error occurred while converting a character string to the internal representation of a FIXEDS number.	VREPLACE DISPLAY	Make sure the data used to update the dialog variable contains valid characters. The valid characters used in updating a dialog variable defined as FIXEDS are +, -, 0−9, and the thousands separator.
805320	An error occurred while converting a character string to the internal representation of a FIXEDU number.	VREPLACE DISPLAY	Make sure the data used to update the dialog variable contains valid characters. The valid characters used in updating a dialog variable defined as FIXEDU are 0−9 and the thousands separator.
805321	An error occurred while converting a character string to the internal representation of a HEX number.	VREPLACE DISPLAY	Make sure the data used to update the dialog variable contains valid characters. The valid characters used in updating a dialog variable defined as HEX are 0−9 and A−F.

Reason Code	Description	Service	Action Required
805322	An error occurred while converting a character string to the internal representation of a PACK number.	VREPLACE DISPLAY	Make sure the data used to update the dialog variable contains valid characters. The valid characters used in updating a dialog variable defined as PACK are +, -, 0–9, and the decimal and thousands separators.
805323	An error occurred while converting a character string to the internal representation of a BIT string.	VREPLACE DISPLAY	Make sure the data used to update the dialog variable contains valid characters. The valid characters used in updating a dialog variable defined as BIT are 0 and 1.
805324	An error occurred while converting a character string to the internal representation of a FLOAT number.	VREPLACE DISPLAY	Make sure the data used to update the dialog variable contains valid characters. The valid characters used in updating a variable defined as FLOAT are +, -, 0–9, and the decimal and thousands separators.
805325	An error occurred while converting a signed number to a dialog variable defined with an unsigned format. The dialog variable must be defined as FIXEDS or FIXED with a length of four to be considered signed.	VREPLACE DISPLAY	Ensure that the character data used to update the dialog variable does not contain a - or + sign.
805326	The variable data contains a hexadecimal digit that is not valid for a packed-decimal number.	VCOPY DISPLAY	Change the value of the dialog variable so that it contains a valid packed-decimal number.
805327	The variable data was too large to fit in the *copy-area*.	VCOPY	Increase the size of the *copy-area*.
805700	A DBCS character was split and data truncation occurred while updating a dialog variable that was defined with a DBCS conversion format.	VREPLACE DISPLAY	Reduce the size of the character string used to update the dialog variable or increase the defined length of the variable being updated. Also, ensure that the length of the variable being updated is an even number.
805701	A DBCS character was split while updating a dialog variable that was defined with a DBCS conversion format.	VREPLACE DISPLAY	Ensure that the length of the character string used to update the dialog variable and the length of the variable being updated are even numbers.

Return Code 12

Reason Code	Description	Service	Action Required
1200100	Variable substitution is not permitted on a DMOPEN call or when the DM communication area is not valid.	DMOPEN	Remove ampersand(&) variable from DMOPEN. Verify that the DM communication area is valid.
1200101	Error converting API buffer to uppercase.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
1200102	Error converting API buffer to uppercase.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
1200103	Error converting API service name in buffer to uppercase.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
1200104	Unexpected parentheses found in buffer.	Any	Verify that parentheses follow keywords as specified.
1200105	Application buffer length greater than 512 bytes.	Any	Correct the buffer length as specified on the ISPCI/ISPCI2 call.
1200106	A parameter that is not valid exists in the application buffer.	Any	Verify buffer content and buffer length.
1200107	Buffer contains more variables or keywords than permitted for service.	Any	Verify buffer content and buffer length.
1200108	The field does not contain valid numeric characters.	Any	Specify numeric characters following keyword.
1200109	A keyword that was misspelled or not valid was found.	Any	Verify buffer content and buffer length.
1200110	The field following a keyword is missing.	Any	Verify keyword and required subfields. Verify buffer length.
1200111	Required field or keyword missing.	Any	Specify all required keywords and name lists. Verify buffer length is correct.
1200112	Required positional parameter missing.	Any	Specify *name-list* immediately after keyword as indicated.
1200113	The number of characters contained in the parameter or variable name was not valid.	Any	Change the parameter name to include the number of allowable characters as specified for a particular service.
1200114	Parenthesized list found where not expected.	Any	Verify service call and keywords.
1200115	Too many names in list following keyword.	Any	Verify list following keyword and length of buffer.

Reason Code	Description	Service	Action Required
1200116	Keyword conflict. Only one choice is permitted, or option is valid only if another field is specified.	Any	Verify service syntax and keyword dependencies.
1200117	Command service not provided by Dialog Manager. Not all services are supported for a Procedures Language application.	Any	Verify service name is a valid DM service.
1200118	End of buffer found or parentheses missing.	Any	Verify parenthesized lists in buffer and buffer length.
1200119	End of buffer found before all fields processed.	Any	Verify keyword subfields and buffer length are correct.
1200120	Subscripted variable not valid for service.	Any	Verify subscript is a valid variable name or within range.
1200121	Service requires an ISPCI2 call, or extra parameters are NULL.	Any	Verify ISPCI/ISPCI2 and additional parameters are valid for the specified service.
1200122	Error during variable substitution. Variable name is not valid.	Any	Verify that the variable name contains valid characters and that the variable is a valid dialog variable for variable substitution to be performed.
1200123	All parameters were processed, but buffer still has more to process.	Any	Verify service syntax and buffer length.
1200124	Extra parameters specified on ISPCI were not valid.	Any	Verify the ISPCI call and the parameters.
1200125	Application buffer size is greater than maximum allowed.	Any	Correct the buffer length parameter.
1200126	First name in buffer does not match table of valid service names.	Any	Verify the first word in the buffer is a valid service name.
1200127	First name in buffer does not have a valid length for a service name.	Any	Verify the first word in the buffer is a valid service name and buffer length is correct.
1200128	Application buffer does not contain keywords or service name. Buffer content not valid.	Any	Verify the first word in the buffer is a valid service name and buffer length is correct.
1200129	*Name-list* found when a single name was expected.	Any	Verify service syntax and buffer length.
1200130	A variable name in the positional parameter field exceeds the allowed length.	Any	Verify syntax of variable names in list and the buffer length.
1200131	The required positional parameter field was not found.	Any	Verify the required *name-list* immediately follows the service name. Verify the buffer length and that parentheses are matched.

Reason Code	Description	Service	Action Required
1200132	The required positional parameter field was not found.	Any	Verify the required *name-list* immediately follows the service name. Verify the buffer length and that parentheses are matched.
1200133	The required positional parameter field was not found.	Any	Verify the required *name-list* immediately follows the service name. Verify the buffer length and that parentheses are matched.
1200134	End of buffer found before end of list. Parentheses may be missing.	Any	Verify that parentheses are matched.
1200135	Keyword not found in keyword name list.	Any	Verify service syntax and buffer length.
1200136	The length of the keyword was not valid.	Any	Verify service syntax and buffer length.
1200137	A particular name to follow a keyword was not found.	Any	Verify service syntax and buffer length.
1200138	Subfield requires numeric characters.	Any	Code numeric characters following specific keywords.
1200139	A required parameter on a subfield is missing.	Any	Verify syntax for keyword subfields.
1200140	Error occurred converting a numeric character string to numeric.	Any	Verify field following keyword contains a valid numeric character string. Verify buffer length and that parentheses are matched.
1200141	A numeric parameter is not in a valid range.	Any	Verify the numeric character string following keyword is in the specified range.
1200502	Command action specified is not supported.	DISPLAY	Verify that the dynamically defined command action is valid.
1200505	An error occurred processing an external command's action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1200508	An error occurred processing an internal command's action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1200520	An error occurred in a user action exit while responding to the CM_INIT_USER_ACTION message.	DISPLAY	Determine the cause of the problem in the user exit and correct the error.
1200521	An error occurred in a user check exit while responding to the CM_INIT_USER_CHECK message.	DISPLAY	Determine the cause of the problem in the user exit and correct the error.

Reason Code	Description	Service	Action Required
1200522	An error occurred in a user control while responding to the CM_INIT_USER_CONTROL message.	DISPLAY	Determine the cause of the problem in the user control and correct the error.
1200523	An error occurred in a user command action exit while responding to the CM_INIT_USER_CMD_ACTION message.	DISPLAY	Determine the cause of the problem in the user exit and correct the error.
1200524	An error occurred in a user translate exit while responding to the CM_INIT_USER_XLATE message.	DISPLAY	Determine the cause of the problem in the user exit and correct the error.
1200525	The application failed to register the *class-name* specified on the ACTION tag.	DISPLAY	Register the user control class after successfully performing the DMOPEN.
1200526	The application failed to register the *class-name* specified on the CHECKI tag.	DISPLAY	Register the user control class after successfully performing the DMOPEN.
1200527	The application failed to register the *class-name* specified on the UC tag.	DISPLAY	Register the user control class after successfully performing the DMOPEN.
1200528	The application failed to register the *class-name* specified on the CMDACT tag.	DISPLAY	Register the user control class after successfully performing the DMOPEN.
1200529	The application failed to register the *class-name* specified on the VARLIST tag.	DISPLAY	Register the user control class after successfully performing the DMOPEN.
1200530	The application failed to register the *class-name* specified on the XLATL tag.	DISPLAY	Register the user control class after successfully performing the DMOPEN.
1200532	The field name specified in the CURSOR parameter on the DISPLAY service call or the PANEL tag is not defined on the panel.	DISPLAY	Specify an input field that exists on the panel.
1200533	The field name specified in the CURSOR parameter on the DISPLAY service call or the PANEL tag cannot accept input.	DISPLAY	Specify an input field.
1200535	An error was reported setting the focus.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1200537	The list field NUMROWS is not valid (that is, the array data does not have at least "NUMROWS" number of elements).	DISPLAY	Correct application logic.

Reason Code	Description	Service	Action Required
1200539	The list field TOPROW tag specifies a row number that is out of range (that is, the top row number is greater than the number of displayable rows).	DISPLAY	Correct application logic.
1200540	Returned when the list field column cannot accept the input focus because it is output-only.	DISPLAY	Specify an input field.
1200542	One or more of the row numbers specified in the modified rows array variable is greater than the number of elements in the list column array variables.	DISPLAY	Make sure the MODVAR values are less than or equal to the number of rows in the list field.
1200543	The selection list NUMROWS is not valid (that is, the array data does not have at least "NUMROWS" number of elements).	DISPLAY	Correct application logic.
1200544	One of the elements in the selection list selection variable is out of the range of valid rows.	DISPLAY	Make sure the SELVAR values are less than or equal to the number of items in the selection list.
1200545	The selection list top row is outside the range of the number of rows.	DISPLAY	Make sure the TOPROW value is less than or equal to the number of items in the selection list.
1200546	The Dialog Manager was unable to load the file containing the fonts for the display of text on a panel.	DISPLAY	Make sure the font files are in the C:\OS2\DLL directory and that the directory is specified in the LIBPATH environment variable of the CONFIG.SYS.
1200547	The text to be specified formats longer vertically than is possible to display in a single information region.	DISPLAY	Break the text to be displayed into several information regions.
1200549	The default command area prompt text could not be obtained.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1200550	An error was returned setting a pool variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1200553	An error occurred in a user variable exit while responding to the CM_INIT_USER_VARIABLE message.	DISPLAY	Determine the cause of the problem in the user exit and correct the error.

Reason Code	Description	Service	Action Required
1200554	Variable services reported an error in response to a request to get a variable from the pool to put into a panel variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1200555	During the display of a panel, a match could not be found for a value entering a list of translation items.	DISPLAY	Check the initial values of the panel variables and make sure they are appropriate for their respective variable classes.
1200556	The length of the variable retrieved from the pools was greater than the maximum length of the panel variable as defined by the VARCLASS tag.	DISPLAY	Ensure that your VDEFINE service calls and VARCLASS declarations are correct.
1200557	The message parameter size of the CM variable is incorrect.	DISPLAY	Set the MessageParmSize value equal to the size of the CM message variable, or ensure that the variable sent matches the appropriate CM message.
1201700	The DM communication area is not valid.	FORCEXIT	Make sure that a valid DMOPEN was performed for this DM communication area.
1201701	Unable to do a FORCEXIT from the Dialog Manager thread.	FORCEXIT	Start a separate thread to use the FORCEXIT service.
1201702	The DM communication area structure does not identify a Dialog Manager thread. This error applies to the FORCEXIT CLEAR service only.	FORCEXIT	Make sure that a valid DMOPEN was performed for this DM communication area.
1201703	An ADDPOP was requested without a prior DISPLAY service call.	ADDPOP	Correct application logic.
1201704	No panel was displayed in the previous pop-up window (that is, two consecutive ADDPOP requests).	ADDPOP	Correct application logic.
1201705	The application has requested that a pop-up window be removed when none has been created.	REMPOP	Correct application logic.
1201708	The application requested that a pop-up window be created before the primary window was created.	ADDPOP	Create the primary window by issuing a DISPLAY call with a non-blank panel name.
1201709	The name specified on the POPLOC parameter is not a field on the currently displayed panel.	ADDPOP	Specify a field name that exists on the currently displayed panel.

Reason Code	Description	Service	Action Required
1201710	The application requested that a new pop-up window be created when no panel had yet been displayed in the current pop-up window.	ADDPOP	Correct application logic.
1201711	Returned when the application specifies a MSGLOC field name that is not defined for the current panel.	DISPLAY	Specify a field name that is defined for the current panel.
1202100	An extended help panel was not defined for the displayed DM application panel.	DISPLAY	Define an extended help panel for the DM application panel and specify its name as the value of the HELP attribute on the PANEL tag.
1202101	The name of the keys help panel to display was not defined using the ZKEYHELP system variable.	DISPLAY	Define the ZKEYHELP variable with the name of the keys help panel to display.
1202138	The Dialog Manager was unable to load the file containing the fonts for the display of a panel.	DISPLAY	Make sure the font files are in the C:\OS2\DLL directory and that the directory is specified in the LIBPATH environment variable of the CONFIG.SYS.
1202139	Unable to load the files containing the fonts used by Dialog Manager.	DMOPEN	Make sure the font files are in the C:\OS2\DLL directory and that the directory is specified in the LIBPATH environment variable of the CONFIG.SYS.
1202901	The application specified UNIQUE on a DMOPEN call and, due to a previous DMOPEN, the UNIQUE parameter is not valid.	DMOPEN	Delete the UNIQUE parameter or change the *application-id* specified.
1202902	The DM communication area is not recognized by Dialog Manager. The corresponding DMOPEN might not have been successful, or the contents of the DM communication area have been corrupted.	DMCLOSE	You must supply the DM communication area of a successful DMOPEN.
1202903	An error occurred starting the DM dynamic trace.	DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1203300	In the application library definition file (xxxxLIB.APP), there is a LIBDEF command that is longer than 512 bytes.	DMOPEN	Correct the LIBDEF command to be shorter than or equal to 512 bytes.
1203301	The LIBDEF service was called with the COND parameter, and a library definition for LIBRARY type already exists. The LIBDEF service call is not processed.	LIBDEF	Check application logic.

Reason Code	Description	Service	Action Required
1203302	The LIBDEF service was called with the COND parameter, and a library definition for HELP type already exists. The LIBDEF service call is not processed.	LIBDEF	Check application logic.
1203303	The requested key mapping list was not found in the currently defined list of libraries.	DISPLAY DMOPEN	Ensure the the LIBDEF service call lists the correct libraries.
1203304	The requested icon was not found in the currently defined list of libraries.	DISPLAY DMOPEN	Ensure the the LIBDEF service call lists the correct libraries.
1203305	The requested message was not found in the currently defined list of libraries.	DISPLAY DMOPEN	Ensure the the LIBDEF service call lists the correct libraries.
1203306	The requested panel was not found in the currently defined list of libraries.	DISPLAY DMOPEN	Ensure the the LIBDEF service call lists the correct libraries.
1203307	The requested command table was not found in the library.	DISPLAY DMOPEN	Check to see if your command table file, xxxxCMDS.DTL, is the correct one.
1203308	At least one LIBDEF command that was not valid was found in the application library definition file, xxxxLIB.APP.	DMOPEN	Edit xxxxLIB.APP and correct the LIBDEF commands that are not valid.
1203309	An unmatched quotation mark was found in the LIBLIST parameter of the LIBDEF command.	DMOPEN LIBDEF	Add another quotation mark to enclose the file name.
1203310	No file names were found in the LIBLIST parameter of the LIBDEF command.	DMOPEN LIBDEF	Add file names to the LIBLIST. If a file name includes commas or blanks, enclose the file name within quotation marks.
1203700	The DISPLAY service call specified a *message-id* that does not exist.	DISPLAY	Specify a *message-id* of a message that is in the application's message library.
1205300	The dialog variable name specified was not valid.	Any	Make sure your dialog variable names conform to the following conventions: • First character a – z or A – Z • Remaining characters a – z, A – Z, 0 – 9, underscore or hyphen • No DBCS characters.

Reason Code	Description	Service	Action Required
1205301	The dialog variable name specified on the the VDEFINE service was not valid.	VDEFINE	Make sure your dialog variable names conform to the following conventions: • First character a – z or A – Z • Remaining characters a – z, A – Z, 0 – 9, underscore or hyphen • No DBCS characters.
1205302	The dialog variable name specified on the the VDELETE service was not valid.	VDELETE	Make sure your dialog variable names conform to the following conventions: • First character a – z or A – Z • Remaining characters a – z, A – Z, 0 – 9, underscore or hyphen • No DBCS characters.
1205303	The dialog variable name specified on the VCOPY service was not valid.	VCOPY	Make sure your dialog variable names conform to the following conventions: • First character a – z or A – Z • Remaining characters a – z, A – Z, 0 – 9, underscore or hyphen • No DBCS characters.
1205304	The dialog variable name specified on the the VREPLACE service was not valid.	VREPLACE	Make sure your dialog variable names conform to the following conventions: • First character a – z or A – Z • Remaining characters a – z, A – Z, 0 – 9, underscore or hyphen • No DBCS characters.
1205305	The dimension parameter specified on the VDEFINE service request was not within the allowable range.	VDEFINE	The *number-of-occurrences* value specified by the DIM keyword must be in the range 1 – 65535.
1205306	A dialog variable being defined does not have the same dimension as the dialog variable whose data is to be copied.	VDEFINE	Change the *number-of-occurrences* value specified by the DIM keyword to match the dimension of the dialog variable whose data is to be copied.
1205307	The value of *n* specified in the PACK(n) format parameter on the VDEFINE service is not within the allowable range.	VDEFINE	The value of *n* must be in the range 0 – 18.

Reason Code	Description	Service	Action Required
1205308	The value of *n* specified in the FLOAT(n) format parameter on the VDEFINE service is not within the allowable range.	VDEFINE	The value of *n* for a short floating point number must be in the range 0 – 25. The value of *n* for a long floating point number must be in the range 0 – 50.
1205309	The length of the PACK number specified on the VDEFINE call is too small to provide the requested number of digits that are to appear to the right of the decimal point.	VDEFINE	Increase the length of the dialog variable.
1205310	The subscript value specified on the variable service request was not valid.	VCOPY VREPLACE	The subscript must be specified as a dialog variable or as a number.
1205311	The subscript was specified as a dialog variable but the variable was not defined.	VCOPY VREPLACE	Use the VDEFINE service to define the dialog variable specified as a subscript.
1205312	The subscript was defined as a dialog variable. The variable was found but had a format that was not valid. If a dialog variable is to be used as a subscript, it must be defined as a signed 4-byte integer.	VCOPY VREPLACE	Use the VDEFINE service to define the dialog variable as FIXED with a length of 4.
1205313	The subscript value was not within the allowable range of 1 – 65535.	VCOPY VREPLACE	Change the value of the subscript, or the value of the dialog variable specified as the subscript, so that it is in the range 1 – 65535.
1205314	A variable service request was issued for a subscripted dialog variable. The variable was not found and was not added. Subscripted dialog variables are not automatically added as implicit variables.	VREPLACE DISPLAY	Use the VDEFINE service with the DIM keyword to define the dialog variable in the dialog variable pool with a *number-of-occurrences* value.
1205315	The subscript specified on the variable service request was not within the dialog variable's defined *number-of-occurrences* range.	VCOPY VREPLACE DISPLAY	Change the value of the subscript so that is within the dialog variable's defined dimension or define the variable with a greater *number-of-occurrences* value.
1205316	A variable service request was made for a subscripted dialog variable. The variable was found in the function pool but was not defined with a *number-of-occurrences* value.	VCOPY VREPLACE DISPLAY	Use the VDEFINE service with the DIM keyword to define the dialog variable in the function pool with a *number-of-occurrences* value.
1205317	The length specified in the *length* parameter of the VREPLACE service was not valid.	VREPLACE	The length of the replace data must be in the range 0 – 32767.

Reason Code	Description	Service	Action Required
1205318	The length specified in the *length* parameter of the VCOPY service was not valid.	VCOPY	The length of the *copy-area* must be in the range 0 – 32767.
1205319	The length specified in the *length* parameter of the VDEFINE service was not within the allowable range for the dialog variable being defined.	VDEFINE	Check the format of the dialog variable being defined and make sure the length specified is within the allowable range.
1205320	An asterisk was specified as a place holder in the name list but a corresponding asterisk was not specified in the format list.	VDEFINE	Change the corresponding format in the format list to an asterisk.
1205321	The number of dialog variable names in the *name-list* does not match the number of format types in the FORMAT list on the VDEFINE call when the *LIST* option was specified.	VDEFINE	Ensure that the number of names specified in the *name-list* matches the number of names specified in the format list.
1206200	The name specified for the Procedures Language *dmcomm-variable* stem is a Procedures Language variable name that is not valid.	Any	Change the name of *dmcomm-variable*. Refer to Procedures Language documentation for requirements.

Return Code 16

Reason Code	Description	Service	Action Required
1600100	The NLS table of keywords and service names was not found.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
1600500	There was not enough memory to allocate the action bar choice or pull-down choice text.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600501	Could not allocate command action buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600502	Could not obtain default command table.	DISPLAY	Call your IBM service representative.
1600504	Could not scroll backward.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600505	Could not scroll forward.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600506	Could not scroll left.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600507	Could not scroll right.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600508	Unable to set list field input field name.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600509	Could not redisplay the panel after executing the CHGDEFS command.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600510	Could not pop a command string from the command stack.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600511	Could not place a command string popped from the command stack in the command entry field.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1600512	Could not update ZVERB.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600513	Could not update ZCMD.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600514	Could not Cancel the panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600515	The DM system commands EXIT, CANCEL, or command action PASSTHRU or SETVERB received an error when trying to store remaining commands for a panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600516	The command area could not complete ENTER processing.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600517	Could not Enter panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600518	Could not Exit panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600519	Could not update ZPANELID.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600520	Could not update ZFKA.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600521	Could not change the current FKA form.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600522	Could not toggle the current panel ID.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1600523	Could not return to the application while processing a PASSTHRU action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600524	Could not return to the application while processing a SETVERB action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600525	Could not push the command string onto the command stack.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600526	Could not refresh the command area with the command string popped from the command stack.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600527	Could not refresh the command available variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600528	Could not refresh the command skip variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600531	Could not resolve a command assigned to a key.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600532	Could not refresh a command action variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600533	An error occurred when passing an external command to the default command table.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600534	An error occurred when passing an internal command to the default command table.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600535	An error occurred when passing a key's command to the default command table.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1600536	Could not process ALIAS.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600537	Could not process an external command.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600538	An error occurred repositioning a field after panel sizing.	DISPLAY	Call your IBM service representative.
1600539	Could not allocate first action parameter buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600540	Could not allocate second action parameter buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600541	Could not allocate SETVAR or TOGVAR variable buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600542	Could not allocate command component's storage.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600543	Could not allocate buffer for shared command information.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600544	Could not allocate buffer for instance command information.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600545	Could not allocate buffer for *application-id*'s command information.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600546	Could not allocate buffer for a saved command string.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600547	Could not allocate buffer for a command stack.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.

Reason Code	Description	Service	Action Required
1600548	Could not allocate buffer for a command stack entry.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600549	An error occurred when passing a command table string to the next command table.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600550	An error occurred when running a SETVAR action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600551	An error occurred updating panel variables when running a SETVAR action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600552	An error occurred when running a TOGVAR action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600553	An error occurred updating panel variables when running a TOGVAR action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600554	Could not obtain the next command table.	DISPLAY	Call your IBM service representative.
1600555	Could not load the default key mapping list.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1600556	Could not load an application's key mapping list.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.

Reason Code	Description	Service	Action Required
1600557	Could not generate the default key mapping list.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1600558	Could not get key mapping list information from the key mapping list.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600560	Could not set the first action parameter.	DISPLAY	Call your IBM service representative.
1600561	Could not set the second action parameter.	DISPLAY	Call your IBM service representative.
1600562	Could not set a variable value while processing a SETVAR or TOGVAR action.	DISPLAY	Call your IBM service representative.
1600563	Could not run TUTORIAL.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600564	Could not load default command table.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1600565	Could not generate the default command table.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1600566	Could not load an application's command table.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1600567	Could not generate an application's command table.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1600568	Could not run the command specified in the RUN action.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600570	Could not perform the action specified in the panel choice.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600571	Could not run a command assigned to a key.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600572	A data value in the list field has failed its variable class operations.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600573	An error occurred updating a list column cell to the variable control.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600574	An error occurred querying the value of a list column cell.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600575	An error occurred when the list field updated the value of a list column cell.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600576	An error was returned to the list field when it requested storage for the list column data.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600577	An error was returned to the list field when it requested storage for the modified rows array.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600578	Presentation Manager reported an error changing the list field entry field text limit.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.

Reason Code	Description	Service	Action Required
1600579	An error was returned to the list field when it requested that it be repositioned (after sizing).	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600580	An error was returned to the list field while changing the values of the modified rows array variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600581	An error was returned to the list field when it updated the modified rows array variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600582	An error was returned to the list field when it requested storage for the translate buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600583	The attempt to change the attribute on a pull-down choice or an action bar choice failed.	DISPLAY	Call your IBM service representative.
1600585	An error occurred setting a panel variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600587	An error occurred getting a panel variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600588	An error occurred getting memory for the variable data buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600589	An error occurred getting a panel variable from a field that had changed.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600590	An error occurred setting a panel variable from a field that had changed.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600591	An error occurred trying to set minimum/maximum size of a field.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600592	An error occurred getting the value of the variable coded on the PARM parameter of the CMDACT tag.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1600593	An error occurred trying to allocate memory to hold the parameters specified on the ACTION tag.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600594	An error occurred trying to allocate memory to hold the parameters specified on the CMDACT tag.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600595	An error occurred trying to allocate memory to hold the parameters specified on the UC tag.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600596	An error occurred while trying to display a message to notify the user that a panel field was not valid.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600597	An error occurred while trying to initialize the validations and translations defined on the panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600598	An error occurred when changing the text of the command entry field.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600599	The key table was not created successfully.	DISPLAY	Call your IBM service representative.
1600600	Presentation Manager reported it could not create the function key area.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem. Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600601	The minimum/maximum size of the divider could not be set.	DISPLAY	Call your IBM service representative.
1600602	The divider could not set its drawing attributes.	DISPLAY	Call your IBM service representative.
1600603	The divider could not set its drawing cursor position.	DISPLAY	Call your IBM service representative.
1600604	Presentation Manager reported an error drawing the divider.	DISPLAY	Call your IBM service representative.

Reason Code	Description	Service	Action Required
1600605	Presentation Manager reported an error setting the focus to a pushbutton, check box, or radio button.	DISPLAY	Call your IBM service representative.
1600606	The area could not determine the minimum and maximum size of its scrollable region.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600607	An error occurred allocating memory for a control.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600608	Unable to paint the text of the information region.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600609	An error occurred while attempting to query for the metrics (characteristics) of the current font in the information region.	DISPLAY	Refer to the OS/2 return code or other DM reason codes for more details.
1600610	An error occurred while attempting to set the attributes (characteristics) of the text to be displayed in the information region.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1600611	Unable to allocate memory for the list of text formatting attributes in the information region.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600612	Unable to allocate memory for the text formatting attributes of the information region.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600613	Unable to allocate the memory for the table that holds the formatting information for the text in the information region.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600614	Unable to allocate the memory for the array of attributes in the formatting table for the text in the information region.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600615	The text attributes generated by the compiler have been corrupted.	DISPLAY	Erase the .DTL file and recompile the .GML file.
1600616	Unable to allocate the memory for the buffer to hold the metrics (characteristics) of the fonts for the display of the text in the information region on a panel.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.

Reason Code	Description	Service	Action Required
1600617	Unable to locate the requested font for the display of the text on a panel.	DISPLAY	Verify that the font is available in a font file. Verify that the font file is in the C:\OS2\DLL directory and that the directory is specified in the LIBPATH environment variable in the CONFIG.SYS file.
1600618	Unable to allocate memory for the array for the left margins at which the text will be formatted in the information region.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600619	The maximum number of fonts has been loaded, and a request has been made to load another font.	DISPLAY	Structure the panel to use fewer fonts. Presentation Manager defines the maximum number of fonts that can be used for an information region (window).
1600620	The system command action specified is not valid.	DISPLAY	Call your IBM service representative.
1600621	No command action specified for command.	DISPLAY	Call your IBM service representative.
1600622	An error occurred during user control processing.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600623	The panel action specified is not supported.	DISPLAY	Call your IBM service representative.
1600624	The variable or value for a SETVAR action was unspecified.	DISPLAY	Call your IBM service representative.
1600625	The variable, value1, or value2 for a TOGVAR action was unspecified.	DISPLAY	Call your IBM service representative.
1600626	Insufficient memory was available to build the region size list.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600628	Could not process the Backtab.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600629	Could not process the Tab.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600630	Could not process Tab or Backtab.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1600631	The command name for a RUN action was unspecified.	DISPLAY	Call your IBM service representative.
1600632	An error occurred trying to update the data field variable value.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600633	Action bar help was requested, and no action bar existed.	DISPLAY	Call your IBM service representative.
1600634	The list field row specified does not exist.	DISPLAY	Check other fields that might share the DATAVAR variable specified on the LSTCOL tag.
1600635	An error was returned to the data field while trying to display the DM message pop-up.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600636	An error occurred trying to allocate memory for the data field.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600637	An error occurred trying to set the text limit of the data field.	DISPLAY	Refer to the OS/2 return code or other DM reason codes for more details.
1600638	Presentation Manager reported an error setting the input focus to an entry field.	DISPLAY	Call your IBM service representative.
1600639	An error was returned to the selection list when updated the selected rows array variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600640	An error occurred querying the value of a list column cell.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600641	Unable to allocate storage for the selection list translation buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600642	An error occurred while translating one of the selection column variable elements.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600643	Unable to insert a new row in the selection list.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.

Reason Code	Description	Service	Action Required
1600644	An error occurred while querying one of the selection list's selected row's variable elements.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600645	Unable to allocate memory for the array of descriptors (IMatch values from the font attributes) used to distinguish the fonts for the information region.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600646	The row number specified for the modified list field row does not exist.	DISPLAY	Check other fields that might share the MODVAR variable specified on the LSTFLD tag.
1600676	An error occurred when allocating storage for a selection field work buffer.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600677	An error occurred when allocating storage for text within the selection field.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600678	Presentation Manager reported an error setting the selection field button text.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600679	Presentation Manager reported an error changing the selection field check value.	DISPLAY	Call your IBM service representative.
1600681	An error occurred when obtaining country information.	DISPLAY	Refer to the OS/2 return code or other DM reason codes for more details.
1600682	An error occurred when creating the panel's accelerator table.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1600683	An error occurred while setting the panel accelerator table.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1600684	The key mapping list does not provide key information.	DISPLAY	Call your IBM service representative.

Reason Code	Description	Service	Action Required
1600687	An error occurred sizing an output-only data field.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600688	Could not load CHGDEFS panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600689	Could not add CHGDEFS pop-up.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600690	Could not display CHGDEFS pop-up.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600691	Could not remove CHGDEFS pop-up.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600692	Could not define one of the CHGDEFS variables.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600693	Could not get one of the CHGDEFS variables.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600694	Could not update one of the CHGDEFS variables.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600695	Could not delete one of the CHGDEFS variables.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600696	An error occurred when the command table was notified of a default action.	DISPLAY	Call your IBM service representative.
1600697	The command table reported an error querying the default action from the current panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1600698	The title bar text could not be set.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1600699	An error was returned to the command table while trying to display a message pop-up.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1600700	An error occurred while allocating storage for the translate and check buffers.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1600701	An error occurred when a command table was requested.	DISPLAY	Call your IBM service representative.
1600702	An error occurred during user action exit processing.	DISPLAY	Determine the cause of the problem in the user exit and correct the error.
1601700	Unable to exit the current panel. The Presentation Manager message queue may be full.	FORCEXIT	Call your IBM service representative.
1601701	Unable to create the primary window icon (when a window is minimized).	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1601702	Unable to retrieve the primary window icon from the library.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1601703	The panel requested could not be retrieved from the library.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1601704	Unable to create the requested panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1601705	An error occurred while entering the panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1601706	The frame could not be created.	DISPLAY DMOPEN	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1601707	An error occurred while updating the panel to be displayed.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1601708	The help information for the panel to be displayed could not be set.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1601709	Unable to allocate panel manager's instance data.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1601710	Unable to create a panel control.	DISPLAY DMOPEN	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1601711	Unable to allocate panel manager shared storage pool.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1601712	Unable to allocate storage for the panel instance data.	DISPLAY DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1601713	An error was returned by GpiQueryTextBox when trying to calculate the length of a string (in pixels) within a panel.	DISPLAY	Call your IBM service representative.
1602101	Unable to save the names of the help libraries that were identified to the Dialog Manager using the LIBDEF service call.	LIBDEF	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.

Reason Code	Description	Service	Action Required
1602102	An error was returned from the OS/2 Information Presentation Facility when the Dialog Manager attempted to change the display of the panel IDs on the help panels. The message sent to cause the error was HM_SET_SHOW_PANEL_ID.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1602103	The Dialog Manager was unable to display the extended help panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602104	Unable to display the keys help panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602105	Unable to verify if the variable's value was defined with the name of the keys help panel to display.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602106	Unable to run the TUTORIAL command from the application command table.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602111	Unable to allocate memory for the help interface component's local instance data.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1602112	Unable to allocate memory for the help interface component's global instance data.	DMOPEN	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1602113	Unable to allocate memory for the help table of the help interface component.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1602114	Unable to allocate memory for the help subtable of the help interface component.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1602116	Unable to initialize communications with the OS/2 Information Presentation Facility.	DMOPEN	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.

Reason Code	Description	Service	Action Required
1602120	Unable to allocate memory for the name of the extended help panel for the panel that is being displayed.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1602121	Unable to inform the OS/2 Information Presentation Facility of who to inform when help is requested on the displayed panel.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1602122	Unable to inform the OS/2 Information Presentation Facility of which Dialog Manager window the help window should be displayed next to. Unable to inform the OS/2 Information Presentation Facility of the Dialog Manager window with which to communicate when errors occur.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1602123	Unable to inform the OS/2 Information Presentation Facility of the names of the help libraries that contain help panels to be displayed when help is selected.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1602125	Unable to inform the OS/2 Information Presentation Facility of the text that should appear in the title bar of the help window.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1602127	An error occurred while requesting that the OS/2 Information Presentation Facility display a help panel.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1602128	Unable to query for the name of the help panel associated with the action bar item from which help was selected.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602129	Unable to query for the name of the help panel associated with the Dialog Manager item from which help was selected.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602130	Unable to display the requested help panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602131	Unable to display the keys help panel when help was requested on a button in the function key area.	DISPLAY	Refer to the OS/2 return code or other DM reason codes for more details.

Reason Code	Description	Service	Action Required
1602132	Unable to display the extended help panel when help was requested on a field that did not have an associated contextual help panel defined.	DISPLAY	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.
1602135	Unable to allocate memory for the names of newly identified help libraries.	LIBDEF	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1602136	Unable to display a message informing the user of an information, warning, or action condition.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602137	Unable to get the names of the help libraries that were identified using the LIBDEF service call.	LIBDEF	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1602140	An error occurred while requesting the display of the Help for help panel from the OS/2 Information Presentation Facility.	DISPLAY	Refer to the OS/2 return code or other DM reason codes for more details.
1602900	An ID that was not valid was passed to set the FSR semaphore.	DMOPEN DMCLOSE	Call your IBM service representative.
1602901	A component failed during a DMCLOSE service.	DMCLOSE	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603300	The application library definition file, xxxxLIB.APP, was found but could not be opened.	DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603301	The application library definition file, xxxxLIB.APP, was opened, but the contents of the file could not be read.	DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603302	Opening a library file failed while processing the request for a key mapping list element.	DISPLAY DMOPEN	Make sure that the library files are in the correct directory, and that the correct library names are specified in the LIBDEF. Refer to the OS/2 return code or other DM reason codes for more details.

Reason Code	Description	Service	Action Required
1603303	Opening a library file failed while processing the request for an icon element.	DISPLAY DMOPEN	Make sure that the library files are in the correct directory, and that the correct library names are specified in the LIBDEF. Refer to the OS/2 return code or other DM reason codes for more details.
1603304	Opening a library file failed while processing the request for a message element.	DISPLAY DMOPEN	Make sure that the library files are in the correct directory, and that the correct library names are specified in the LIBDEF. Refer to the OS/2 return code or other DM reason codes for more details.
1603305	Opening a library file failed while processing the request for an application panel.	DISPLAY DMOPEN	Make sure that the library files are in the correct directory, and that the correct library names are specified in the LIBDEF. Refer to the OS/2 return code or other DM reason codes for more details.
1603306	Opening a library file failed while processing the the request for a command table element.	DISPLAY DMOPEN	Make sure that the library files are in the correct directory. Refer to the OS/2 return code or other DM reason codes for more details.
1603307	A directory in the library could not be read while processing the request for a key mapping list element.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603308	A directory in the library could not be read while processing the request for an icon element.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603309	A directory in the library could not be read while processing the request for a message element.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603310	A directory in the library could not be read while processing the request for an application panel element.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603311	A directory in the library could not be read while processing the request for a command table element.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.

Reason Code	Description	Service	Action Required
1603312	The requested key mapping list element was found in the library, but it could not be read.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603313	The requested icon element was found in the library, but it could not be read.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603314	The requested message element was found in the library, but it could not be read.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603315	The requested panel element was found in the library, but it could not be read.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603316	The requested command table element was found in the library, but it could not be read.	DISPLAY DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603317	An error occurred as a result of a LIBDEF update.	LIBDEF	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603318	An attempt to allocate memory during library services initialization failed.	DMOPEN	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603319	An attempt to allocate memory for a buffer to read a library definition file failed.	DMOPEN	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603320	An attempt to allocate memory for the LIBDEF *lib-list* failed.	DMOPEN LIBDEF	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603700	An error occurred when attempting to display the message pop-up window.	DISPLAY	Call your IBM service representative.
1603701	The ADDPOP request for the message panel failed.	ADDPOP	Call your IBM service representative.
1603702	The creation of the message panel failed.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603703	A request for memory to concatenate the *message-id* and the MSGTEXT failed.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.

Reason Code	Description	Service	Action Required
1603704	Request to load the resource for the message panel failed.	DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1603705	The message text in the message pop-up could not be set.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603706	The attempt to redisplay the panel that was active before the message panel was displayed failed.	DISPLAY	Call your IBM service representative.
1603707	An error occurred while attempting to remove the message pop-up from the screen.	DISPLAY	Call your IBM service representative.
1603708	The message pop-up text could not be set.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603709	An error occurred while setting the return code for a severe error into the return code system variable.	DISPLAY	Call your IBM service representative.
1603710	Message services could not allocate its shareable instance data.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603711	An error occurred during the allocation of message services session instance data.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603712	The storage needed to store the text for Dialog Manager's critical messages could not be allocated.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603713	An error occurred attempting to load the message resource file.	DMOPEN DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.

Reason Code	Description	Service	Action Required
1603714	The message member could not be created.	DMOPEN DISPLAY	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1603715	An error occurred while trying to get a message resource.	DMOPEN DISPLAY	Refer to the OS/2 return code or other DM reason codes for more details.
1603716	An error occurred retrieving message information.	DMOPEN DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603717	The storage needed to hold the the message text requested could not be allocated.	DMOPEN DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603718	An error occurred requesting the value of variables in message text.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603719	An error occurred during the translation of a message variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603720	The panel used for displaying Dialog Manager severe error messages could not be created.	DMOPEN	Contact your IBM service representative.
1603721	An error occurred while loading the dialog box resource used for creating the severe error message panel.	DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
1603722	The accelerator table used for the severe error message panel could not be created.	DMOPEN	This error could have occurred due to lack of available memory or window handles from Presentation Manager. Reducing the complexity of the application panels or the number of pop-up windows displayed by the application at one time may eliminate this problem.
1603723	An error occurred setting the accelerator table to the severe error message panel.	DMOPEN	Refer to the Presentation Manager ERRORID retrieved using the WinGetLastError call for more information.

Reason Code	Description	Service	Action Required
1603724	The buffer used for the variable substitution of variable in message text could not be allocated.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603725	The buffer used to facilitate VARCLASS operations on variables in message text could not be allocated.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603726	An error occurred updating the message panel.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
1603727	An error occurred while setting the window title of the severe error dialog box.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1603728	An error occurred setting the button text for the Enter button in the severe error dialog box.	DISPLAY	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1604100	An error occurred in storage manager.	Any	Call your IBM service representative.
1604101	No memory is available to be allocated.	Any	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
1604102	Storage manager internal pool control block could not be allocated.	Any	Call your IBM service representative.
1604103	Storage could not be sub-allocated.	Any	Call your IBM service representative.
1604104	All allocated segments were not freed.	Any	Call your IBM service representative.
1604105	Unable to free LST storage of block that was combined.	Any	Call your IBM service representative.
1604106	Unable to delete a storage pool.	Any	Call your IBM service representative.
1604107	Unable to free space control block.	Any	Call your IBM service representative.
1604108	Unable to expand a storage pool.	Any	Call your IBM service representative.
1604109	Unable to free selector chain link.	Any	Call your IBM service representative.
1604110	Unable to free pools storage block.	Any	Call your IBM service representative.
1604111	Unable to free pools LST storage.	Any	Call your IBM service representative.
1604112	Unable to free a previously allocated segment.	Any	Call your IBM service representative.

Reason Code	Description	Service	Action Required
1604113	Unable to allocate storage for an LST.	Any	Call your IBM service representative.
1604114	Unable to allocate a pools storage block.	Any	Call your IBM service representative.
1604115	DosSubSet failed to initialize segment.	Any	Call your IBM service representative.
1604116	Requested storage length was less than or equal to zero.	Any	Call your IBM service representative.
1604117	Storage to be freed goes past the end of a pools storage block. Caused by storage length that was longer than requested or a storage pointer other than what storage manager returned on the storage request.	Any	Call your IBM service representative.
1604118	Storage request greater than 65488 bytes.	Any	Call your IBM service representative.
1604119	Storage could not be allocated for the selector chain link.	Any	Call your IBM service representative.
1604120	Segment to be freed was not found in the selector chain.	Any	Call your IBM service representative.
1604121	Unable to share initial segment.	Any	Call your IBM service representative.
1604122	Unable to share a storage segment.	Any	Call your IBM service representative.
1604123	Storage to be freed was not allocated from the specified pool.	Any	Call your IBM service representative.
1604124	Unable to free the specified storage.	Any	Call your IBM service representative.
1605701	DosGetCtryInfo failed trying to retrieve country information. Possible causes include corrupted system country information file.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
1605702	DosCaseMap failed trying to convert string to uppercase. Possible causes include corrupted system country information file.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
1606200	A Procedures Language dialog attempted to run in a non-Presentation Manager session.	Any	Ensure that the program is being run properly. See Chapter 10, "Compiling, Linking, and Running Dialog Manager Applications" on page 10-1 for Procedures Language requirements.
1606201	An error occurred when trying to access the Procedures Language variable pool.	Any	Ensure that the Procedures Language files are installed properly.

Return Code 20

Reason Code	Description	Service	Action Required
2000506	Dialog Manager class registration failed.	DMOPEN	Check installation of Dialog Manager.
2000515	Variable services reported an error when a variable was moved from the panel into the dialog variable pool.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
2000516	Variable services reported an error in response to a request to get a variable from the pools to put into a panel variable.	DISPLAY	Refer to the other DM reason codes in the Error Information section of the DM communication area for more details.
2002900	An error occurred trying to create the space for the storage manager to use.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002901	An error occurred trying to create the pool for Dialog Manager.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002902	An error occurred trying to get storage for instance manager.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002903	Dialog Manager failed to complete its initialization.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002904	An error occurred obtaining access to the segments for the current process.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
2002906	An error occurred trying to get storage for the *instance-id* entry in the instance manger.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002907	DM communication area was corrupted or the instance manager table was corrupted while adding an entry to the table.	DMOPEN DMCLOSE	Call your IBM service representative.
2002908	DM communication area was corrupted or the instance manager table was corrupted while deleting an entry on a DMCLOSE.	DMCLOSE	Call your IBM service representative.
2002909	An error occurred trying to get storage for instance manager process ID entry.	Any	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.

Reason Code	Description	Service	Action Required
2002910	Unable to give a process access to a segment.	Any	Refer to the OS/2 return code or other DM reason codes for more details.
2002911	The DM communication area is not valid. The DM communication area is not valid if its address is zero or if the contents has been corrupted since the last DMOPEN. A DM communication area is no longer valid after a DMCLOSE.	Any except DMOPEN.	Either a DMOPEN service call needs to be added to obtain a valid DM communication area, or the DM communication area has been corrupted.
2002912	Unable to fold the *application-id* to uppercase.	DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
2002913	Presentation Manager could not be initialized.	DMOPEN	Reinstall PM or fix the CONFIG.SYS. Make sure the Presentation Manager files are installed properly and the paths to the DM dynamic link libraries are in the LIBPATH environment variable in the CONFIG.SYS.
2002914	Presentation Manager could not create a message queue.	DMOPEN	Re-install PM or fix the CONFIG.SYS. Make sure the Presentation Manager files are installed properly and the paths to the DM dynamic link libraries are in the LIBPATH environment variable in the CONFIG.SYS. Make sure that your application does not create a message queue. Verify that your module definition file contains the WINDOWAPI keyword on the NAME statement.
2002915	An error occurred trying to allocate storage for the data entry for the instance manager.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002916	Instance manager could not perform a DMOPEN.	DMOPEN	Call your IBM service representative.
2002917	DosExitList failed while trying to add an entry during DMOPEN.	DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
2002918	An error occurred trying to allocate storage for the anchor block handle during a DMOPEN.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.

Reason Code	Description	Service	Action Required
2002919	An error occurred trying to allocate storage for the *instance-id* during a DMOPEN.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002920	A component in Dialog Manager failed on a DMOPEN service.	DMOPEN	Refer to the OS/2 return code or other DM reason codes for more details.
2002921	Unable to access the NLS codepage tables during a DMOPEN.	DMOPEN	Call your IBM service representative.
2002922	Too many parameters are missing on the DMOPEN. If it is the first DMOPEN, the APPLID parameter must be specified.	DMOPEN	Add the appropriate parameters on the DMOPEN service call. Correct the application logic.
2002923	The APPLID contains characters that are not valid.	DMOPEN	Change the APPLID to only contain valid characters. Correct the application logic.
2002924	The *instance-id* specified on the DMOPEN cannot be used in this thread.	DMOPEN	Use the *instance-id* specified in the DM communication area returned on the DMOPEN in the current thread. Correct the application logic.
2002925	The Dialog Manager does not recognize the *instance-id*.	DMOPEN	Use the *instance-id* specified in the DM communication area returned on the DMOPEN in the current thread. Correct the application logic.
2002926	An *instance-id* already exists on this thread. The DMOPEN call must share the *instance-id*.	DMOPEN	Add the INSTID parameter to the DMOPEN service call. Correct the application logic.
2002927	Presentation Manager cannot create an object window.	DMOPEN	This error occurred due to lack of available memory. Adding memory or closing other applications may solve this problem.
2002928	An error occurred doing a DosExitList call to remove an entry.	DMCLOSE	Call your IBM service representative.
2002929	An error occurred trying to delete an *instance-id* during a DMCLOSE.	DMCLOSE	Call your IBM service representative.
2002930	An error occurred trying to register the DM classes during a DMOPEN call.	DMOPEN	Make sure the DM files are installed properly and the paths to the files are in the CONFIG.SYS file.
2004100	Unable to add storage to an available storage list.	Any	Call your IBM service representative.
2004101	Unable to allocate storage for the pools first storage block.	Any	Call your IBM service representative.
2004102	Unable to allocate pool's control block	Any	Call your IBM service representative.

Reason Code	Description	Service	Action Required
2004103	Unable to allocate pool's storage element.	Any	Call your IBM service representative.
2004104	Unable to allocate initial storage space.	Any	Call your IBM service representative.
2004105	Could not create default storage pool.	Any	Call your IBM service representative.
2004106	Unable to create internal pool.	Any	Call your IBM service representative.
2004107	Unable to create selector chain link pool.	Any	Call your IBM service representative.
2004108	Unable to free the LST to the internal storage pool.	Any	Call your IBM service representative.

Glossary

This glossary defines terms as they are used by the Dialog Manager. For additional definitions, refer to the *Dictionary of Computing*, SC20-1699.

A

action. One of the defined tasks that an application performs. Actions modify the properties of an object or manipulate the object in some way. Users may choose an action in several ways: by typing a command, pressing a function key, selecting the action from an action bar pull-down, or by selecting a pushbutton.

action bar. The area at the top of a primary window that contains keywords that give users access to actions available in that window. After users select a choice in the action bar, a pull-down extension appears from the action bar. Also known as *menu bar*.

action bar pull-down. An extension of the action bar that displays a list of actions related to a selected choice in the action bar. After users select a choice in the action bar, the pull-down appears. Additional pop-up windows can appear from pull-down choices. Also known as *menu*.

action message. Information that appears in a message pop-up. It tells users that an exception condition has occurred. Users must perform an action to correct the situation. Compare with *information message* and *warning message*.

application. A collection of software components that people use to perform specific types of work on a computer.

application ID. A 4-character identifier that you specify on the DMOPEN call. The *application-id* identifies the command table and application library definition file.

assignment list. A list that contains values assigned to one variable based on the content of another variable during panel processing.

attribute. A keyword, and its associated value, that specifies a particular characteristic (for example the name, size, or type) of items such as a field, panel, or command table.

available choice. An item that the current state of the application allows users to select.

C

cascade. To arrange windows and icons so that the top left corner of each is below and to the right of the one it overlaps. The windows appear like index cards stacked one behind the other.

character. A symbol used in printing and viewing. For example, a letter of the alphabet, a numeral, a punctuation mark, or any other symbol that represents information.

character unit. A unit of measure of horizontal and vertical space that is dependent on the system or display device. The average character unit is the width of a lowercase "o."

check box. A choice containing a square box with descriptive text beside it. It acts like a switch. An "X" appears in the check box to show that a choice is selected. Check boxes can be used alone or grouped in related sets so users can choose one or more choices. Check boxes are provided through Dialog Manager's multiple-choice selection field.

check item. A DTL tag that defines a single test of an input value. This test applies to the value the user enters for a data field or a list field.

check list. A list of check items.

client area. The part of a window inside the border that is below the action bar. It is the user's work space, where users type information and choose selections from selection fields. It can contain the following panel elements: top instructions, panel body, bottom instructions, and the command area.

command. The name and parameters associated with an action that can be performed by a program. Commands are associated with pull-downs, selection fields, function keys, and the command area.

command area. A panel element composed of a command prompt and a command entry field.

command area prefix. A displayed symbol that indicates where the user enters commands. The default is ===>.

command entry field. The part of the command area into which users type commands.

command field prompt. A word or words that identify a command entry field. The default is "Command" or its translated equivalent.

command procedure. A program written in Procedures Language 2/REXX.

contextual help. Specific information about the item that the cursor is on. This help information is *contextual* because it describes information about a specific item as it is currently used. The information provided is specific to the meaning of the item within the work session. Also known as *field-level help*. Contrast with *extended help*.

cursor. A visual cue that shows users the current position of the keyboard input focus. The keyboard cursors are the *selection cursor* and the *text cursor*.

D

data field. A field that can accept input or display output or both. A data field also includes the prompt text and description text. See also *entry field* and *output field*.

DBCS. See *double-byte character set*.

declaration subset. A part of the DOCTYPE declaration statement where entity and parameter entity declarations are made.

de-emphasize. To use shading, color, or half-toning to indicate that an item on a panel, such as a pull-down choice, is unavailable. Also known as *graying*.

dialog. The communication between a person and a computer. Within the Dialog Manager, a dialog starts when a DM application issues the DMOPEN service to establish communication with the Dialog Manager. All subsequent service calls, to display a panel or to define a variable, belong to the same dialog. A dialog ends when the DM application issues the DMCLOSE call or when the application ends.

dialog element. A part of a DM application defined with the Dialog Tag Language (DTL). Dialog elements include panels, message members, key mapping lists, icons, and an application command table. You define the dialog elements with DTL tags in a source file. Dialog elements are later compiled and stored on disk.

dialog interface. An SAA definition of services and other facilities that can be used in writing DM applications.

Dialog Manager. A set of services and facilities that complement those of the underlying operating system to provide a highly productive method of developing an application to carry on an interactive dialog with a user.

Dialog Tag Language (DTL). A markup language used to define DTL elements. DTL is based on the Standard Generalized Markup Language (SGML).

dialog variable. A variable that is used to communicate data between the DM application and Dialog Manager.

dialog variable pool. A collection of dialog variables associated with the DM communication area. Only a program having access to the DM communication area used at the time the pool was created can access the variables in the pool.

divider. A solid or blank line that provides users with a visual distinction between two adjacent portions of an application panel. Also known as *separator*.

DM application. An application comprised of programs, panels, and messages that use the services and facilities provided by the Dialog Manager.

DM communication area. An area of program storage that the Dialog Manager uses to communicate status to, and process information for, the DM application.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages, such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

DTL. See *Dialog Tag Language*.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

entity. A collection of characters that can be referenced as a unit in a DTL markup source file. Entities may be contained within the source file they are referenced from or within external files.

entry field. The portion of a data field into which users can type. See also *data field*.

explicitly defined variable. A dialog variable that is defined when a program uses the VDEFINE service with that variable name.

extended help. Information about the contents of the application window from which users requested help. Also known as *general help*. Contrast with *contextual help*.

F

field. A visual panel element. Fields can display variable data or accept user input. The types of fields that Dialog Manager supports are: data fields, list fields, selection fields, and selection lists.

field prompt. Descriptive, static text that identifies a selection field or an entry field.

function key. A key that causes a specified sequence of operations to be performed when it is pressed. An example is F1 for Help.

function key area. The area at the bottom of a panel that contains the function key assignments in pushbuttons.

H

help. Information about a specific field, an application window, or a pop-up window.

help for help. Information provided to users about how to use help.

help index. An index, with search capability, of all the help information in the application.

help panel. A panel that contains information to assist users. It is displayed in a help window.

help window. A Common User Access-defined window that contains information to assist users.

I

icon. A pictorial representation of an object or a selection choice. Icons can represent objects that users want to work on or actions that users want to perform. A unique icon also represents the application when it is minimized.

implicitly defined variable. A variable that the Dialog Manager defines when a Dialog Manager service refers to a dialog variable name that is not found in the dialog variable pool, or when a Dialog Manager service must store data in a dialog variable that does not already exist in the dialog variable pool.

information message. Information that appears in a message pop-up. It tells users that a function is performing normally or has performed normally. Compare with *action message* and *warning message*.

Information Presentation Facility. An OS/2 programming tool that can be used to develop an SAA Common User Access-conforming help interface.

input field. See *data field*.

input focus. The area of a window where keyboard interaction is possible.

instance identifier. An identifier the Dialog Manager assigns to a dialog when the dialog starts. Dialog Manager uses the instance ID to manage the resources associated with the dialog.

K

keys help. A listing of an application's keys and their assigned functions.

L

list box. A field that contains scrollable choices from which users can select. Also known as *selection list*.

M

markup. Text that is added to document data in order to convey information about it. There are three types of markup that the DTL uses: tags, references, and markup declarations.

markup declaration. Markup that controls how other markup of a document is to be interpreted, for example document type and entity declarations.

markup language. A set of characters, conventions, and rules to control the interpretation of document data. The Dialog Tag Language is a markup language.

message. Information not requested by users but shown by the computer in response to a user action or an internal process. Messages appear in a message pop-up.

message member. A dialog element that contains messages grouped together for use by the application.

message pop-up. A type of window that displays messages to users. Also known as *message box*.

message type. Indicates the severity of different types of messages. A message can be one of three types: action, warning, or information. An action message has the highest severity, a warning message has the next-highest, and an information message has the lowest.

mixed data. Data that contains both single-byte (SBCS) and double-byte characters (DBCS).

mnemonic. A single character, within the text of a choice, identified by an underscore. When users type a choice's mnemonic, that choice is selected.

modal. A method of operation that requires the user to complete interaction with a pop-up window before continuing to work in the window from which the pop-up was displayed.

mouse. An instrument used to move the pointer on the screen.

N

name list. A list of dialog variable names passed as a single parameter in a dialog service call.

O

output field. A type of data field that displays variable data. Users cannot enter data into an output field. See also *data field*.

P

panel. A particular arrangement of user interface components, such as an action bar, entry fields, selection fields, and list fields presented to users in a window.

panel body. A panel element that serves as the main work area of the panel.

The panel body of an application panel can contain: data fields, selection fields, selection lists, list fields, and information regions. An application panel can contain a scrollable area if defined with an AREA tag.

The panel-body of a help panel contains help text and is scrollable.

panel definition. A description of the contents and characteristics of a panel. Thus, a panel definition is the application developer's mechanism for predefining the format of information to be presented to users in a window.

panel element. A part or parts of a panel definition. A single panel element (such as an action bar) can contain other panel elements, such as action bar choices. Panel elements can also be non-visual, such as a region.

panel ID. A panel element located to the left of the window title that identifies that particular panel. If the panel ID is visible, it appears in the left margin of the title bar and is separated from the window title with a hyphen (-).

parameter. A keyword whose value specifies characteristics for the Dialog Manager service with which it is associated.

parameter entity. An entity that is referenced from a markup declaration.

pointer. The symbol displayed on the screen that is moved by a pointing device, such as a mouse. It is also used to point at the objects users want to select and at actions they choose to perform on the selected objects.

pointing device. An instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pop-up window. A movable window, fixed in size, that extends the user's dialog with another window. Therefore, pop-up windows are associated with other underlying windows and appear when the application wants to extend the dialog in the underlying window. Users must finish interacting with the pop-up window before continuing with the dialog in the related window.

primary window. The window in which the main dialog between users and the application takes place. In a multitasking environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward.

program variable. A named changeable value that can exist only within a program. Its value cannot be obtained or used when the program that contains it is no longer active.

pull-down. See *action bar pull-down*.

pushbutton. A rounded-corner rectangle with text inside. Pushbuttons are used in pop-up windows for actions that occur immediately when the pushbutton is selected. Pushbuttons are provided through Dialog Manager's immediate-action selection field.

R

radio button. A choice containing a circle with descriptive text beside it. Radio buttons are combined to show users a fixed set of choices that are mutually exclusive. These fields must contain at least two choices, one of which is always selected. The circle is partially filled when a choice is selected. Radio buttons are provided through Dialog Manager's single-choice selection field.

reason code. A field in the DM communication area. The reason code further defines the return code from a dialog service call.

reference phrase help. User-selectable words or phrases within a help window for which additional information is available.

return code. A numeric code indicating the result of an operation. The return code is in the first 4 bytes of the DM communication area.

S

SBCS. See *single-byte character set*.

screen. The physical surface of a display device upon which information is presented to users.

scroll bar. A window component associated with a scrollable area, that provides users a visual cue that more information is available. Users scroll the information in the window by interacting with the scroll bar.

selection field. A field that contains a set of selectable choices.

selection list. A field that contains scrollable choices from which users can select. Also known as *list box*.

shift-in (SI) character. In EBCDIC, a hexadecimal X'0F' character that indicates the end of a DBCS character string in mixed data.

shift-out (SO) character. In EBCDIC, a hexadecimal X'0E' character that indicates the beginning of a DBCS character string in mixed data.

single-byte character set (SBCS). A set of characters in which each character is represented by one byte.

stacked VDEFINE. Two or more VDEFINE calls without corresponding VDELETE calls using the same variable name.

system extensions. Services and facilities that provide system-dependent function. System extensions are not part of the Systems Application Architecture.

system variable. Variables provided by the dialog interface that communicate special information between the DM application and the Dialog Manager. System variable names always begin with the character Z.

T

tag. One or more characters attached to a set of data that defines the formatting or other characteristics of the set, including its identification.

tutorial. An application-defined action that provides access from the current window to a tutorial, if the application has one.

U

unavailable choice. An item that the current state of the application does not allow users to select because of some condition in the application. Unavailable choices are displayed with unavailable emphasis. Contrast with *available choice*.

unavailable emphasis. A visual cue that shows users which items in a list of choices are currently unavailable for selection. See also *de-emphasize*

W

warning message. Information that appears in a message pop-up. It tells users that a potentially undesirable situation could occur. Users only need to respond to the message to continue. Corrective action may be required later to avoid an error situation.

window. An area of the screen with visible boundaries through which information is displayed. A window can be smaller than or equal in size to the screen. Windows can overlap on the screen and give the appearance of one window being on top of another.

window title. The area in the title bar that identifies the information in the panel.

Special Characters

% notation. A special notation used to distinguish a variable name from a specific value when both can be assigned to a tag attribute. Precede the variable name with the percent (%) sign. When the % sign is used, the attribute value must be enclosed in quotes.

Index

D

I

K

L

M

N

O

P

R

S

T

U

V

W

Y

Z

Special Characters

IBM United Kingdom
International Products Limited
PO Box 41, North Harbour
Portsmouth, PO6 3AU
England

Printed in Denmark

Scanprint, Jyllands-Posten A/S, Viby J.